НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету Інформаційних технологій	**Завідувач кафедри** Комп'ютерних систем, мереж та кібербезпеки
_____Ігор БОЛБОТ (підпис)	_____Дмитро КАСАТКІН (підпис)
«___»_____2026 р.	«___»_____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: «Розробка вебпанелі моніторингу серверів і пристроїв»

Спеціальність F7 «Комп'ютерна інженерія »

Освітньо-професійна програма «Комп'ютерна інженерія »

Гарант освітньої програми канд. фіз.-мат. наук, доцент	_____ (підпис)	**Євгеній НІКІТЕНКО**
Керівник бакалаврської кваліфікаційної роботи канд. пед. наук, доцент	_____ (підпис)	**Дмитро КАСАТКІН**
Виконав	_____ (підпис)	**Дмитро ХОМУТОВСЬКИЙ**

КИЇВ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
канд. пед. наук, доцент_____Дмитро КАСАТКІН
«____»_____20____ р.

З А В Д А Н Н Я

ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

ЗДОБУВАЧУ

Хомутовському Дмитру Івановичу
(прізвище, ім'я, по батькові)

Спеціальність «Комп'ютерна інженерія »

Освітньо-професійна програма «Комп'ютерна інженерія »

Тема кваліфікаційної бакалаврської роботи
«Розробка вебпанелі моніторингу серверів і пристроїв »

затверджена наказом ректора НУБіП України від «10»___12_____2025 р. №___3072 «С»___
Термін подання завершеної роботи на кафедру «2»___06_____2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи

Перелік питань, що підлягають дослідженню:

Перелік графічного матеріалу (за необхідності).

Дата видачі завдання "__10__"_____12_____2025 р.

Керівник бакалаврської кваліфікаційної роботи канд. пед. наук, доцент	_____ (підпис)	**Дмитро КАСАТКІН**
Завдання взяв до виконання	_____ (підпис)	**Дмитро ХОМУТОВСЬКИЙ**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	10.02.2026 р.	Виконано
2	Проектування системи	02.03.2026 р.	Виконано
3	Реалізація системи	20.03.2026 р.	Виконано
4	Тестування системи	10.05.2026 р.	Виконано
5	Оформлення пояснювальної записки	20.05.2026 р.	Виконано
6	Оформлення графічного матеріалу	01.06.2026 р.	Виконано

Студент _____

(підпис) (ініціали та прізвище)

Керівник проекту (роботи) _____

(підпис) (ініціали та прізвище)

Реферат

Пояснювальна записка до дипломного проєкту «Розробка вебпанелі моніторингу серверів і пристроїв» містить _97_ с., _47_ рис., _9_ табл., 33 літературні джерела.

У роботі описано розробку системи моніторингу серверів і клієнтських пристроїв для контролю основних показників стану IT-інфраструктури. Система побудована на клієнт-серверній архітектурі та включає Ubuntu Server, API на Node.js та Express, базу даних MySQL, клієнтський агент на PowerShell і вебпанель для відображення стану пристроїв.

Розглянуто актуальність моніторингу комп'ютерної інфраструктури, проаналізовано засоби Zabbix, Prometheus, Grafana, Nagios та PRTG. На основі аналізу обґрунтовано створення власної спрощеної вебпанелі для навчального й демонстраційного стенду.

У роботі спроєктовано архітектуру системи, структуру бази даних, алгоритм роботи серверного API та клієнтського агента. Система збирає такі показники: завантаження процесора, використання оперативної пам'яті, заповнення дискового простору, IP-адресу, статус пристрою та час останнього оновлення.

Практична реалізація виконана у середовищі Oracle VM VirtualBox на базі трьох віртуальних машин: Ubuntu Server, Windows 10 і Windows Server. Клієнтські пристрої передають дані на сервер через HTTP POST-запити, після чого інформація зберігається в базі даних і відображається у вебпанелі.

У результаті створено працездатний програмний стенд системи моніторингу серверів і пристроїв. Тестування підтвердило коректну роботу клієнтського агента, серверного API, бази даних MySQL і вебпанелі. Система може використовуватися для навчальних, лабораторних і демонстраційних цілей.

Ключові слова: ВЕБПАНЕЛЬ, МОНІТОРИНГ, СЕРВЕР, API, NODE.JS,

EXPRESS, MYSQL, POWERSHELL, UBUNTU SERVER, WINDOWS, КЛІЄНТСЬКИЙ АГЕНТ, ІТ-ІНФРАСТРУКТУРА.

Abstract

The explanatory note to the diploma project “Development of a Web Panel for Monitoring Servers and Devices” contains 97 pages, 47 figures, 9 tables, and 33 references.

The paper describes the development of a system for monitoring servers and client devices to control the main indicators of IT infrastructure status. The system is based on a client-server architecture and includes Ubuntu Server, an API developed with Node.js and Express, a MySQL database, a PowerShell client agent, and a web panel for displaying device status.

The relevance of computer infrastructure monitoring is considered, and monitoring tools such as Zabbix, Prometheus, Grafana, Nagios, and PRTG are analyzed. Based on the analysis, the development of a custom simplified web panel for an educational and demonstration stand is substantiated.

The system architecture, database structure, server API operation algorithm, and client agent algorithm are designed in the paper. The system collects the following indicators: CPU load, RAM usage, disk space usage, IP address, device status, and last update time.

The practical implementation was carried out in the Oracle VM VirtualBox environment using three virtual machines: Ubuntu Server, Windows 10, and Windows Server. Client devices transmit data to the server via HTTP POST requests, after which the information is stored in the database and displayed in the web panel.

As a result, a functional software stand for monitoring servers and devices was created. Testing confirmed the correct operation of the client agent, server API, MySQL database, and web panel. The system can be used for educational, laboratory, and demonstration purposes.

Keywords: WEB PANEL, MONITORING, SERVER, API, NODE.JS, EXPRESS, MYSQL, POWERSHELL, UBUNTU SERVER, WINDOWS, CLIENT AGENT, IT INFRASTRUCTURE.

ЗМІСТ

ВСТУП

У сучасних умовах функціонування підприємств, навчальних установ, державних організацій та приватних компаній інформаційна інфраструктура є одним із ключових елементів забезпечення безперервності основних бізнес-процесів. Сервери, робочі станції, мережеві служби, бази даних, вебзастосунки та інші цифрові ресурси забезпечують зберігання, оброблення, передавання й візуалізацію інформації. Порушення їхньої працездатності може призвести до простоїв, втрати даних, погіршення якості обслуговування користувачів, а також до збільшення витрат на відновлення роботи системи.

Одним із важливих напрямів адміністрування інформаційних систем є моніторинг стану серверів і клієнтських пристроїв. Моніторинг дає змогу своєчасно виявляти перевантаження центрального процесора, нестачу оперативної пам’яті, заповнення дискового простору, втрату мережевої доступності або інші ознаки потенційної несправності. У рекомендаціях NIST щодо безперервного моніторингу інформаційної безпеки підкреслюється необхідність постійного отримання актуальної інформації про стан інформаційних систем, вразливості та загрози для підтримки прийняття управлінських рішень [1].

Актуальність теми дипломної роботи зумовлена тим, що навіть у невеликих локальних мережах адміністратору необхідно мати централізований інструмент для спостереження за станом декількох пристроїв. Перевірка кожного комп’ютера окремо є неефективною, потребує багато часу та не дає змоги оперативно реагувати на зміни. Тому доцільним є створення системи, у якій клієнтські пристрої автоматично передають показники свого стану на сервер, сервер зберігає ці дані в базі даних, а користувач переглядає їх через зручну вебпанель.

У межах цієї роботи розглядається розроблення вебпанелі моніторингу серверів і пристроїв, яка працює на основі клієнт-серверної архітектури. Згідно з наданими матеріалами проєкту, система має включати Ubuntu Server як сервер збору та оброблення даних, Windows 10 і Windows Server як клієнтські пристрої, базу даних для збереження показників, API для приймання даних і вебпанель для

відображення інформації про стан пристроїв . Основними показниками, які повинна відображати система, є завантаження CPU, RAM, Disk, IP-адреса пристрою, онлайн/офлайн-статус, час останньої активності та, за можливості, графіки зміни показників у часі.

Окреме значення у системах моніторингу має збір і зберігання журналів подій. У документації NIST щодо керування журналами комп'ютерної безпеки зазначено, що журнали подій є важливим джерелом інформації для виявлення інцидентів, аналізу стану системи та підтримки процесів адміністрування [2]. Хоча в межах цієї дипломної роботи основна увага приділяється не повноцінному журналюванню подій безпеки, а моніторингу ресурсів пристроїв, сама ідея централізованого збору даних відповідає загальним принципам керування станом інформаційної інфраструктури.

Метою дипломної роботи є розроблення вебпанелі моніторингу серверів і клієнтських пристроїв, яка забезпечує автоматизований збір, збереження та відображення основних показників стану комп'ютерних систем у локальній мережі.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. проаналізувати предметну область моніторингу серверів і клієнтських пристроїв;
2. розглянути основні показники стану комп'ютерних систем, які доцільно контролювати;
3. дослідити принципи побудови клієнт-серверних систем моніторингу;
4. спроєктувати архітектуру системи моніторингу;
5. розробити структуру бази даних для збереження інформації про пристрої;
6. реалізувати серверне API для приймання даних від клієнтських агентів;
7. реалізувати клієнтський агент для передавання показників із Windows-пристроїв;
8. створити вебпанель для відображення стану пристроїв;
9. розгорнути систему у віртуальному середовищі;

10. виконати тестування роботи системи та проаналізувати отримані результати.

Об’єктом дослідження є процес моніторингу стану серверів і клієнтських пристроїв у локальній інформаційній інфраструктурі.

Предметом дослідження є програмні засоби збору, передавання, збереження та візуалізації показників стану комп’ютерних систем.

Практичне значення роботи полягає у створенні працездатного програмного стенду, який може бути використаний для демонстрації принципів моніторингу IT-інфраструктури. Такий стенд є корисним для навчальних, лабораторних і практичних цілей, оскільки дозволяє показати повний цикл роботи системи: від збору показників на клієнтському пристрої до їх передавання на сервер, запису в базу даних і відображення у вебінтерфейсі.

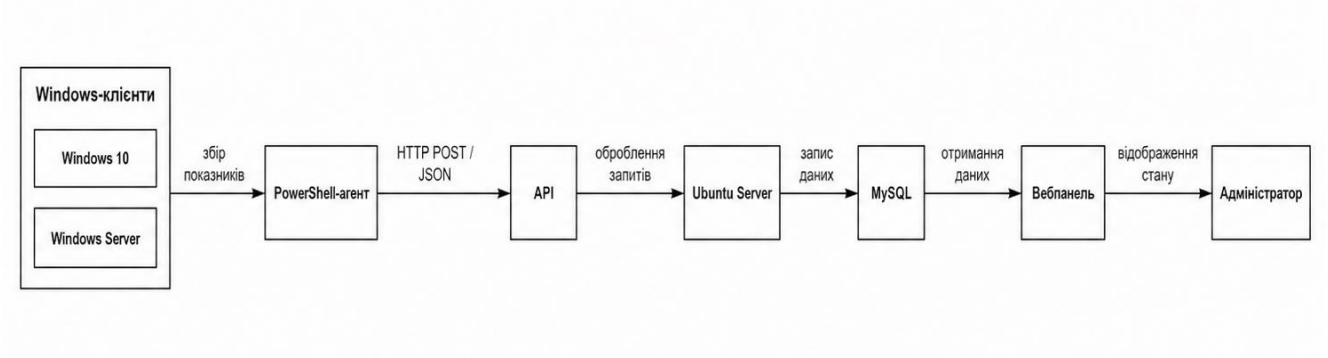

Рисунок 1 — Узагальнена схема роботи системи моніторингу серверів і пристроїв

Джерело: розроблено автором

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАСОБІВ МОНІТОРИНГУ ІТ-ІНФРАСТРУКТУРИ

1.1 Значення моніторингу серверів і клієнтських пристроїв

Моніторинг інформаційної інфраструктури — це процес постійного або періодичного спостереження за станом апаратних, програмних і мережевих компонентів з метою своєчасного виявлення відхилень, несправностей, перевантажень або інших подій, які можуть вплинути на стабільність роботи системи. У практичному розумінні моніторинг охоплює збір показників продуктивності, перевірку доступності пристроїв, контроль використання ресурсів, фіксацію часу останньої активності та формування повідомлень про проблемні стани.

Для серверів моніторинг має особливо важливе значення, оскільки вони забезпечують роботу служб, від яких залежать інші користувачі та пристрої. Якщо сервер бази даних, файловий сервер, вебсервер або сервер прикладного програмного забезпечення стає недоступним, це може призвести до зупинення роботи цілої групи користувачів. Саме тому сервери потребують постійного контролю основних параметрів: завантаження процесора, використання оперативної пам’яті, стану дискового простору, мережевої доступності та активності системних служб.

Клієнтські пристрої також потребують моніторингу, особливо якщо вони виконують важливі робочі функції або є частиною навчального, офісного чи виробничого середовища. Наприклад, якщо комп’ютер користувача втрачає зв’язок із мережею, має критично заповнений диск або працює з постійним перевантаженням процесора, це може впливати на продуктивність працівника та загальну якість роботи інформаційної системи.

У сучасних системах моніторингу часто використовується принцип централізованого збору даних, коли інформація з багатьох пристроїв надходить до одного сервера. Такий підхід дозволяє адміністратору переглядати стан усієї інфраструктури з одного інтерфейсу. Подібну логіку використовують поширені

системи моніторингу, зокрема Prometheus, який збирає та зберігає метрики як часові ряди, тобто значення показників разом із часовими мітками [3].

Іншим прикладом є Zabbix — система моніторингу, у якій агенти можуть встановлюватися на цільові вузли для контролю локальних ресурсів і передавання зібраних даних на сервер моніторингу [4]. Такий підхід є близьким до концепції, що реалізується в межах цієї дипломної роботи: на клієнтських Windows-пристроях працює агент, який збирає дані про стан системи та передає їх на сервер, де вони зберігаються й відображаються у вебпанелі.

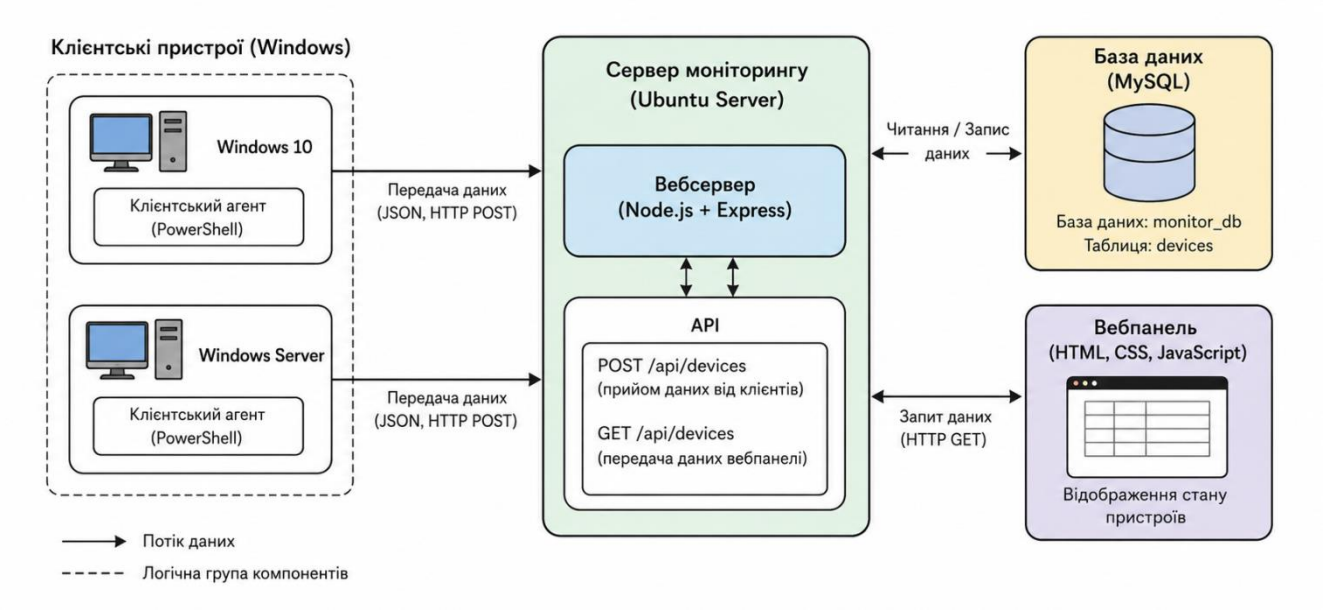

Рисунок 1.1 — Принцип централізованого моніторингу пристроїв

Джерело: розроблено автором

1.2 Основні показники стану комп’ютерних систем

Для оцінювання стану комп’ютерної системи необхідно визначити набір показників, які найбільш точно характеризують її працездатність. У межах розроблюваної системи до таких показників належать завантаження центрального процесора, використання оперативної пам’яті, заповнення дискового простору, IP-адреса пристрою, статус доступності та час останнього оновлення даних.

Завантаження центрального процесора є одним із базових показників продуктивності комп’ютерної системи. Якщо процесор тривалий час працює з

високим рівнем завантаження, це може свідчити про виконання ресурсоємних процесів, помилки в роботі програмного забезпечення або недостатню продуктивність обладнання для поточних задач. У системі моніторингу такий показник дає змогу швидко визначити пристрої, які працюють у режимі перевантаження.

Оперативна пам’ять впливає на здатність системи одночасно виконувати декілька процесів. Нестача RAM може призводити до сповільнення роботи операційної системи, активного використання файлу підкачки та погіршення реакції програм. Тому контроль використання оперативної пам’яті є важливою частиною моніторингу як серверів, так і клієнтських пристроїв.

Дисковий простір є критичним ресурсом для стабільної роботи системи. Заповнення диска може призвести до неможливості запису журналів, оновлень, тимчасових файлів або даних користувачів. Для серверів це може бути особливо небезпечно, оскільки нестача вільного простору здатна порушити роботу баз даних, вебслужб або інших сервісів. Саме тому в системі передбачено контроль показника Disk.

IP-адреса пристрою використовується для ідентифікації вузла в мережі. Її відображення у вебпанелі дозволяє адміністратору розуміти, з якого саме пристрою надходять дані, а також спрощує діагностику мережевих проблем. У наданих матеріалах зазначено, що під час розгортання стенду всі віртуальні машини мають перебувати в одній підмережі, а серверна IP-адреса використовується клієнтськими агентами для передавання даних через API .

Статус доступності пристрою є одним із найважливіших елементів вебпанелі моніторингу. Якщо пристрій регулярно надсилає дані, він може вважатися активним або таким, що перебуває в онлайн-стані. Якщо ж протягом певного часу оновлення від нього не надходять, система може позначити його як неактивний або офлайн. Такий механізм дозволяє адміністратору швидко виявляти пристрої, які втратили зв’язок із сервером.

Час останньої активності використовується для визначення актуальності отриманих даних. Навіть якщо пристрій раніше передавав показники,

адміністратору важливо знати, коли саме було отримано останнє оновлення. Це дає змогу відрізнити актуальний стан системи від застарілої інформації.

Таблиця 1.1 — Основні показники моніторингу комп'ютерних систем

Показник	Призначення	Практичне значення
CPU	Визначення рівня завантаження процесора	Дозволяє виявити перевантажені пристрої
RAM	Контроль використання оперативної пам'яті	Дає змогу оцінити нестачу пам'яті
Disk	Контроль заповнення дискового простору	Допомагає запобігти збоям через нестачу місця
IP-адреса	Ідентифікація пристрою в мережі	Спрощує адміністрування й діагностику
Статус	Визначення онлайн/офлайн-стану	Дозволяє швидко знайти недоступні вузли
Last update	Час останнього оновлення даних	Показує актуальність отриманої інформації

Джерело: розроблено автором

1.3 Аналіз існуючих засобів моніторингу

Для обґрунтування доцільності розроблення власної вебпанелі моніторингу серверів і пристроїв необхідно розглянути наявні програмні рішення, які використовуються для контролю стану IT-інфраструктури. До найбільш відомих засобів моніторингу належать Zabbix, Prometheus, Grafana, Nagios та PRTG Network Monitor. Кожен із цих інструментів має власну сферу застосування, переваги, обмеження та особливості впровадження.

Zabbix є комплексною системою моніторингу, яка дозволяє контролювати сервери, мережеве обладнання, віртуальні машини, служби, бази даних і прикладні системи. Однією з особливостей Zabbix є використання агентів, які встановлюються на контрольовані вузли та передають інформацію про стан системи на сервер моніторингу [4]. Такий підхід є ефективним для корпоративних середовищ, оскільки дає змогу централізовано збирати дані з великої кількості пристроїв. Проте для навчального або демонстраційного стенду повноцінне розгортання Zabbix може бути надлишковим, оскільки система має велику

кількість функцій, потребує налаштування шаблонів, хостів, тригерів і правил оповіщення.

Prometheus є системою моніторингу та збирання метрик, яка зберігає дані у вигляді часових рядів. Основна ідея Prometheus полягає у періодичному отриманні метрик із цільових систем і подальшому аналізі цих даних за допомогою спеціальної мови запитів PromQL [3]. Prometheus добре підходить для хмарних, контейнеризованих і мікросервісних середовищ, де необхідно контролювати велику кількість сервісів. Водночас для невеликого локального стенду, що складається з Ubuntu Server і двох Windows-клієнтів, використання Prometheus може ускладнити проєкт через потребу в додатковому налаштуванні експортерів, правил збору метрик та інтеграції з інструментами візуалізації.

Grafana є інструментом для побудови інформаційних панелей і візуалізації даних. В офіційній документації зазначено, що Grafana підтримує роботу з багатьма джерелами даних і дозволяє візуалізувати інформацію у вигляді панелей, графіків та правил оповіщення [5]. На відміну від систем, які самостійно збирають метрики, Grafana переважно виконує роль інтерфейсу візуалізації. Її часто використовують разом із Prometheus, InfluxDB, Loki або іншими джерелами даних. Для цієї дипломної роботи ідея Grafana є важливою з погляду принципів побудови вебпанелі, однак розроблення власного інтерфейсу дозволяє краще показати логіку роботи системи, структуру API, взаємодію з базою даних і механізм оновлення даних у браузері.

Nagios є одним із відомих засобів моніторингу IT-інфраструктури. Згідно з офіційною документацією Nagios, система може використовуватися для моніторингу критичних компонентів інфраструктури, зокрема системних метрик, мережевих протоколів, застосунків, сервісів, серверів і мережевої інфраструктури [6]. Nagios має розвинену систему перевірок і сповіщень, але його налаштування часто потребує детальної конфігурації об’єктів моніторингу, служб, команд і плагінів. Для великих або критичних інфраструктур це є перевагою, однак для навчального проєкту така складність може відволікати від головної мети — демонстрації базових принципів створення власної системи моніторингу.

PRTG Network Monitor є комерційною системою моніторингу, яка використовує сенсорну модель. У документації Paessler зазначено, що сенсори застосовуються для контролю окремих параметрів пристроїв і сервісів, наприклад мережевого трафіку, SNMP-показників або інших вимірюваних характеристик [7]. Також PRTG підтримує налаштування повідомлень, які можуть спрацьовувати при зміні статусу сенсора або перевищенні заданих меж [8]. Перевагою такого підходу є зручність для адміністратора та наявність готового вебінтерфейсу. Недоліком для дипломного проєкту є те, що значна частина функціональності вже реалізована у готовому продукті, тому використання PRTG не дозволяє повною мірою продемонструвати власну розробку API, бази даних і вебпанелі.

Порівняння існуючих засобів моніторингу показує, що більшість із них орієнтована на професійне або корпоративне використання. Вони мають широкі можливості, але разом із цим потребують часу на вивчення, розгортання та налаштування. У межах дипломної роботи доцільніше створити спрощену систему, яка реалізує основні принципи моніторингу: збір даних із клієнтських пристроїв, передавання їх через API, збереження у базі даних і відображення у вебпанелі.

Таблиця 1.2 — Порівняльна характеристика засобів моніторингу IT-інфраструктури

Засіб моніторингу	Основне призначення	Переваги	Обмеження для цієї роботи
Zabbix	Комплексний моніторинг серверів, служб і мережевого обладнання	Підтримка агентів, тригерів, шаблонів, оповіщень	Надлишкова складність для невеликого навчального стенду
Prometheus	Збір і збереження метрик у вигляді часових рядів	Добре підходить для хмарних і мікросервісних систем	Потребує експортерів і додаткового налаштування
Grafana	Візуалізація даних і побудова дашбордів	Зручні графіки, підтримка багатьох джерел даних	Не є повноцінною системою збору метрик без джерела даних

Продовження таблиці 1.2

Nagios	Контроль доступності служб, серверів і мережевих компонентів	Розвинена система перевірок і сповіщень	Складність конфігурації для навчального проєкту
PRTG	Комерційний моніторинг на основі сенсорів	Готовий вебінтерфейс, сенсори, повідомлення	Значна частина функцій уже реалізована готовим продуктом
Власна вебпанель	Навчальна система збору й відображення базових метрик	Повний контроль над API, БД, агентом і вебінтерфейсом	Менше функцій порівняно з професійними рішеннями

Джерело: розроблено автором

На основі проведеного аналізу можна зробити висновок, що розроблення власної вебпанелі є доцільним у межах дипломної роботи, оскільки дозволяє продемонструвати не лише кінцевий результат, а й увесь процес побудови системи моніторингу. Власна реалізація дає змогу показати принципи клієнт-серверної взаємодії, організацію API, структуру бази даних, роботу клієнтського агента та логіку формування вебінтерфейсу.

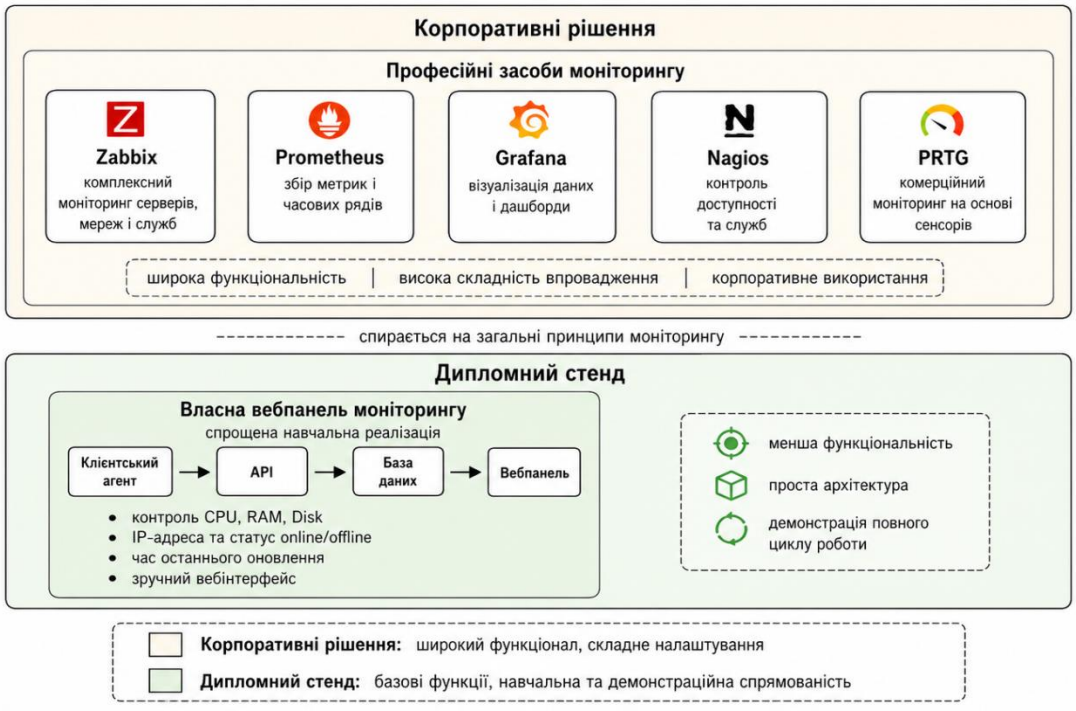

Рисунок 1.2 — Місце власної вебпанелі серед засобів моніторингу

Джерело: розроблено автором

1.4 Обґрунтування вибору власної вебпанелі моніторингу

Розроблення власної системи моніторингу у межах дипломної роботи має декілька важливих переваг. По-перше, воно дозволяє не лише використати готовий програмний продукт, а й самостійно реалізувати основні компоненти системи: клієнтський агент, серверне API, базу даних і вебінтерфейс. Це дає змогу глибше розкрити тему дипломної роботи, оскільки увага приділяється не тільки експлуатації системи, а й її архітектурі, логіці роботи та програмній реалізації.

По-друге, власна вебпанель може бути адаптована саме під поставлену задачу. У цьому проєкті немає потреби реалізовувати весь функціонал великих корпоративних систем. Основною метою є створення зрозумілої та працездатної системи, яка відображає ключові показники стану пристроїв: завантаження процесора, використання оперативної пам’яті, заповнення диска, IP-адресу, статус доступності та час останнього оновлення. Саме такі вимоги визначено у наданому описі проєкту, де зазначено, що система має включати вебпанель, сервер збору даних, клієнтів і обмін даними між ними .

По-третє, власна реалізація дозволяє краще контролювати структуру даних. У наданих матеріалах запропоновано створити таблицю devices, яка містить поля для назви пристрою, IP-адреси, показників CPU, RAM, Disk, статусу та часу останнього оновлення . Така структура є простою, але достатньою для демонстрації основного принципу роботи системи моніторингу. Вона дозволяє зберігати актуальні показники пристроїв і відображати їх у вебінтерфейсі.

По-четверте, використання власного API дає змогу показати механізм взаємодії між клієнтськими пристроями та сервером. У наданих матеріалах передбачено, що клієнтський агент на Windows-пристрої збирає показники за допомогою PowerShell і передає їх на сервер через HTTP-запит . Такий підхід є зрозумілим, відносно простим у реалізації та добре підходить для навчального стенду.

По-п’яте, розроблення власної вебпанелі дає змогу продемонструвати практичні навички роботи з вебтехнологіями. Інтерфейс може містити таблицю

пристроїв, кольорове позначення статусів, автоматичне оновлення даних, а також графіки зміни показників у часі. Це відповідає функціональним вимогам, зазначеним у матеріалах проєкту, де серед бажаних можливостей наведено показ CPU, RAM, Disk, IP-адрес, онлайн/офлайн-статусу, часу останньої активності та графіків .

Таким чином, розроблення власної вебпанелі моніторингу є обґрунтованим рішенням для дипломної роботи. Воно дозволяє поєднати теоретичний аналіз предметної області з практичною реалізацією програмного стенду, який демонструє повний цикл роботи системи моніторингу.

1.5 Постановка задачі

На основі аналізу предметної області та існуючих засобів моніторингу сформульовано задачу дипломної роботи: необхідно розробити вебпанель моніторингу серверів і клієнтських пристроїв, яка забезпечує збір, передавання, збереження та відображення основних показників стану комп’ютерних систем у локальній мережі.

Система повинна складатися з таких основних компонентів (рис. 1.3):

- серверної частини, розгорнутої на Ubuntu Server;
- бази даних для збереження інформації про пристрої;
- API для приймання даних від клієнтів;
- клієнтського агента для Windows-пристроїв;
- вебпанелі для перегляду стану пристроїв;
- віртуального лабораторного середовища для тестування роботи системи.

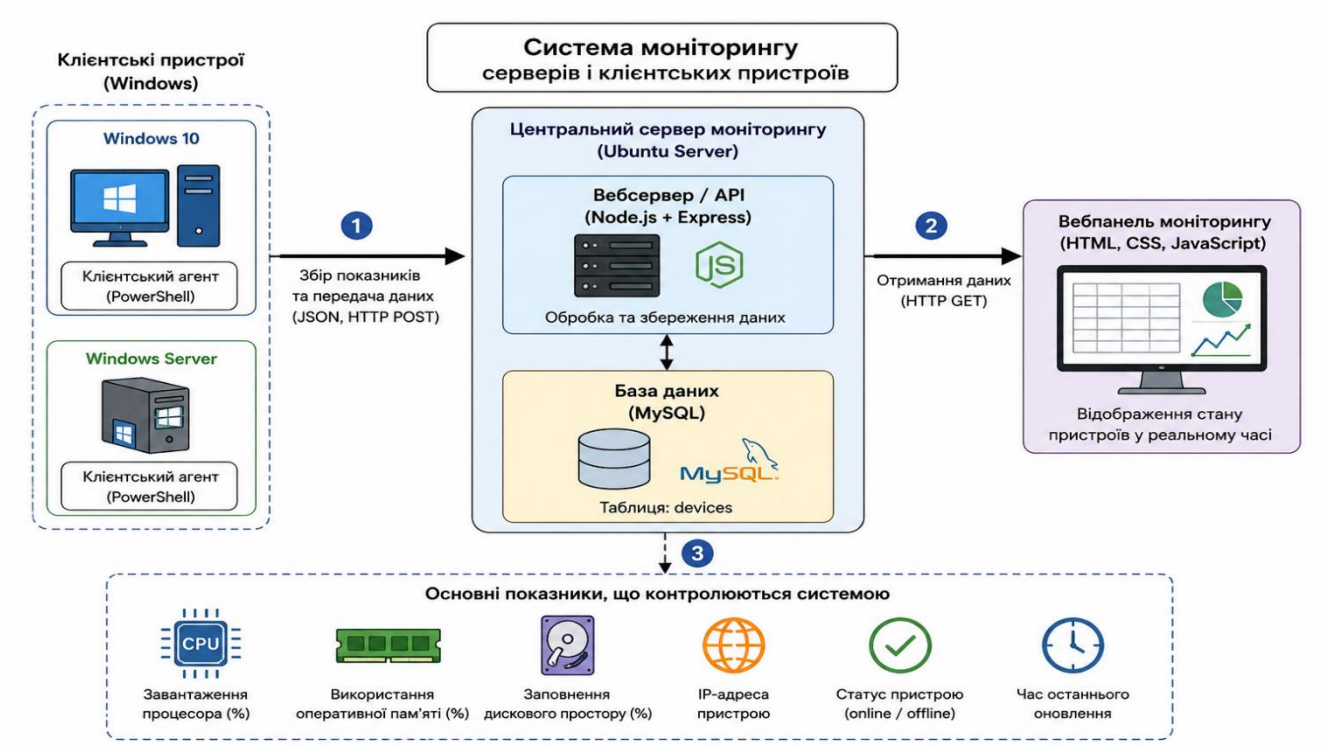

Рисунок 1.3 — Структура компонентів системи моніторингу

Джерело: розроблено автором

Серверна частина повинна приймати дані від клієнтів, перевіряти коректність запитів, записувати або оновлювати інформацію в базі даних і надавати вебпанелі актуальні дані для відображення. У межах реалізації доцільно використати Node.js, Express і MySQL, оскільки така комбінація дозволяє швидко створити HTTP API, організувати оброблення JSON-даних і забезпечити взаємодію з реляційною базою даних.

Клієнтський агент повинен працювати на Windows 10 і Windows Server. Його завданням є отримання локальних показників системи, формування HTTP-запиту та передавання даних на сервер. У наданій інструкції з розгортання стенду зазначено, що на кожній Windows-машині використовується файл monitor.ps1, у якому потрібно вказати актуальну IP-адресу Ubuntu Server для передавання даних на API .

Вебпанель повинна відображати список пристроїв, їхні IP-адреси, показники CPU, RAM і Disk, поточний статус і час останньої активності. Для зручності користувача доцільно реалізувати автоматичне оновлення даних без ручного

перезавантаження сторінки. Це дозволить адміністратору бачити актуальний стан пристроїв у режимі, наближеному до реального часу.

Основними вимогами до системи є:

- централізований збір даних із декількох клієнтських пристроїв;
- передавання даних через HTTP API;
- збереження показників у базі даних;
- відображення інформації у вебпанелі;
- можливість визначення онлайн/офлайн-статусу пристрою;
- простота розгортання у віртуальному середовищі;
- можливість подальшого розширення функціоналу.

Практична реалізація має бути виконана у вигляді віртуального стенду. Згідно з наданими матеріалами, для цього використовуються три віртуальні машини: Ubuntu Server як серверна частина, Windows 10 як клієнтський пристрій і Windows Server як другий клієнтський пристрій . Усі машини повинні бути об’єднані в одну мережу, що забезпечує можливість передавання даних між клієнтами та сервером.

1.6 Висновки до розділу 1

У першому розділі було розглянуто значення моніторингу серверів і клієнтських пристроїв для забезпечення стабільної роботи інформаційної інфраструктури. Визначено, що контроль показників CPU, RAM, Disk, IP-адреси, статусу доступності та часу останнього оновлення дозволяє адміністратору своєчасно виявляти проблемні стани пристроїв і реагувати на них.

Проаналізовано існуючі засоби моніторингу, зокрема Zabbix, Prometheus, Grafana, Nagios і PRTG Network Monitor. Встановлено, що ці системи мають широкі функціональні можливості й активно використовуються в професійному середовищі, однак для навчального дипломного проєкту вони можуть бути надлишковими або недостатньо зручними для демонстрації власної програмної реалізації.

Обґрунтовано доцільність створення власної вебпанелі моніторингу. Такий підхід дозволяє показати повний цикл роботи системи: збір показників на клієнтському пристрої, передавання їх через API, збереження у базі даних і відображення у вебінтерфейсі. Сформульовано основні функціональні вимоги до системи та визначено її ключові компоненти: Ubuntu Server, Node.js API, MySQL, Windows-клієнти, PowerShell-агент і вебпанель.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ СЕРВЕРІВ І ПРИСТРОЇВ

2.1 Загальна архітектура системи

Проєктування системи моніторингу серверів і пристроїв передбачає визначення її основних компонентів, способів взаємодії між ними, структури передавання даних і логіки відображення результатів у вебінтерфейсі. Оскільки система має працювати в локальній мережі та забезпечувати централізоване відображення стану декількох пристроїв, доцільним є використання клієнт-серверної архітектури.

У межах цієї дипломної роботи система моніторингу складається з трьох основних рівнів: клієнтського рівня, серверного рівня та рівня візуалізації. Клієнтський рівень представлений Windows-пристроями, на яких працює агент збору даних. Серверний рівень реалізується на базі Ubuntu Server і включає API, базу даних та серверну логіку оброблення запитів. Рівень візуалізації представлений вебпанеллю, через яку адміністратор переглядає інформацію про стан пристроїв.

Згідно з матеріалами практичної частини, у фінальному варіанті система повинна включати вебпанель, сервер збору даних на Ubuntu Server, клієнти на Windows 10 або Windows Server, а також обмін даними між ними. Основними функціями системи є показ завантаження CPU, RAM, Disk, відображення IP-адрес, онлайн/офлайн-статусу, часу останньої активності та графіків за можливості .

Загальна логіка роботи системи (рис. 2.1) може бути подана у вигляді послідовності: клієнтський пристрій збирає локальні показники, формує HTTP-запит і передає його на серверне API; сервер приймає дані, перевіряє їх, записує або оновлює інформацію в базі даних; вебпанель отримує актуальні дані з сервера та відображає їх адміністратору.

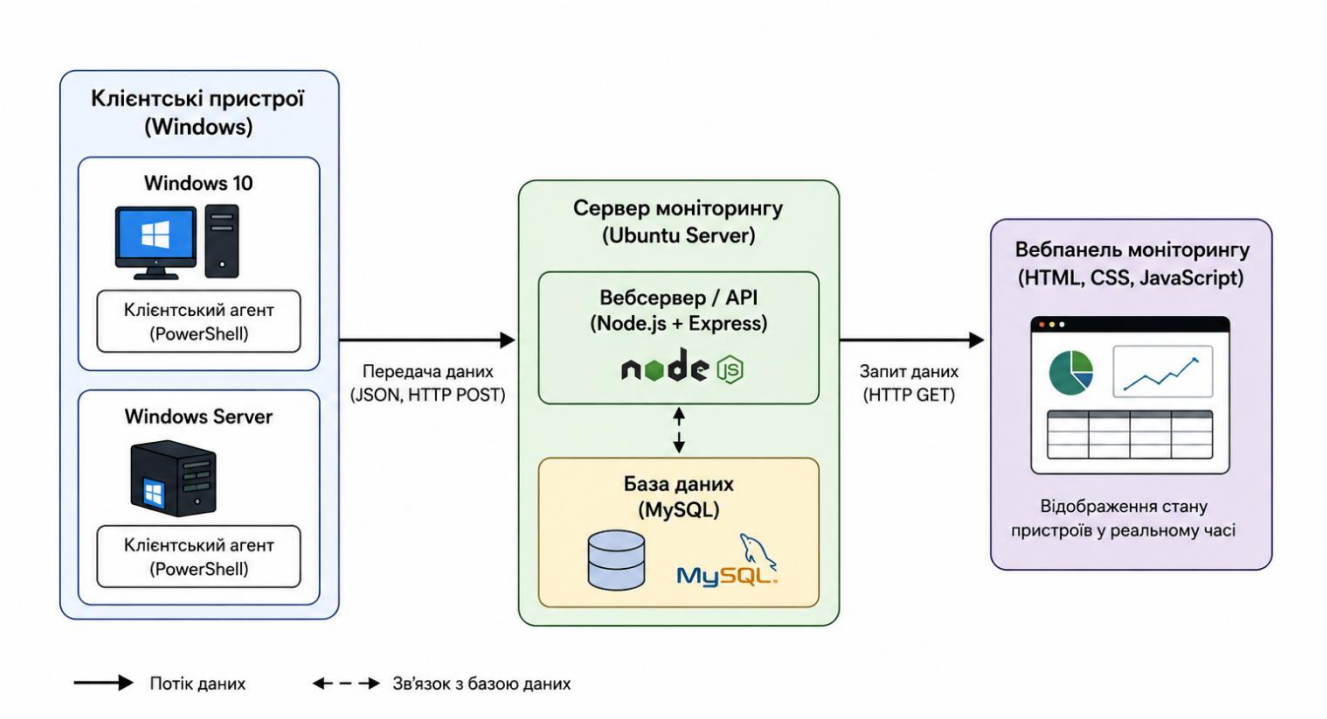

Рисунок 2.1 — Загальна архітектура системи моніторингу серверів і пристроїв

Джерело: розроблено автором

Під час проєктування системи важливо забезпечити простоту, зрозумілість і можливість подальшого розширення. Саме тому було обрано модульну структуру, у якій кожен компонент виконує окрему функцію. Клієнтський агент відповідає тільки за збір і передавання даних. API відповідає за приймання запитів і взаємодію з базою даних. База даних зберігає інформацію про пристрої. Вебпанель відповідає за відображення даних у зручній формі.

Такий підхід дає змогу змінювати або доповнювати окремі частини системи без повної переробки всієї архітектури. Наприклад, у майбутньому можна додати нові типи клієнтських агентів для Linux-пристроїв, реалізувати авторизацію користувачів, додати журнал подій, розширити структуру бази даних або інтегрувати графіки зміни показників у часі.

2.2 Обґрунтування вибору клієнт-серверної моделі

Клієнт-серверна модель є доцільною для систем моніторингу, оскільки вона дозволяє розділити функції збору даних, оброблення інформації, збереження результатів і візуалізації. У такій моделі клієнти виконують роль джерел даних, а сервер — роль центрального вузла, який приймає, обробляє та надає інформацію для перегляду.

Для розроблюваної системи клієнтами виступають Windows 10 і Windows Server. Вони передають інформацію про свій стан на сервер. Сервером виступає Ubuntu Server, на якому розміщено API, базу даних і вебпанель. Така структура відповідає архітектурі, наведений у матеріалах проєкту: **Windows-клієнти → API → Ubuntu сервер → база даних → вебпанель** .

Перевагою клієнт-серверної моделі є централізація управління даними. Адміністратору не потрібно окремо перевіряти кожний пристрій, оскільки всі показники надходять до єдиного сервера. Це спрощує контроль інфраструктури, зменшує час пошуку проблем і підвищує зручність експлуатації системи.

Іншою перевагою є можливість масштабування. Якщо в майбутньому потрібно буде додати нові пристрої, достатньо встановити або налаштувати клієнтський агент на цих пристроях і вказати адресу серверного API. Серверна частина при цьому може залишатися незмінною або потребувати лише незначного розширення.

Клієнт-серверна модель (рис. 2.2) також дає змогу реалізувати єдину точку зберігання даних. Усі показники пристроїв записуються в базу даних, що дозволяє уникнути розпорошення інформації між різними комп’ютерами. У подальшому це може бути використано для формування історії змін, графіків, звітів або повідомлень про аварійні стани.

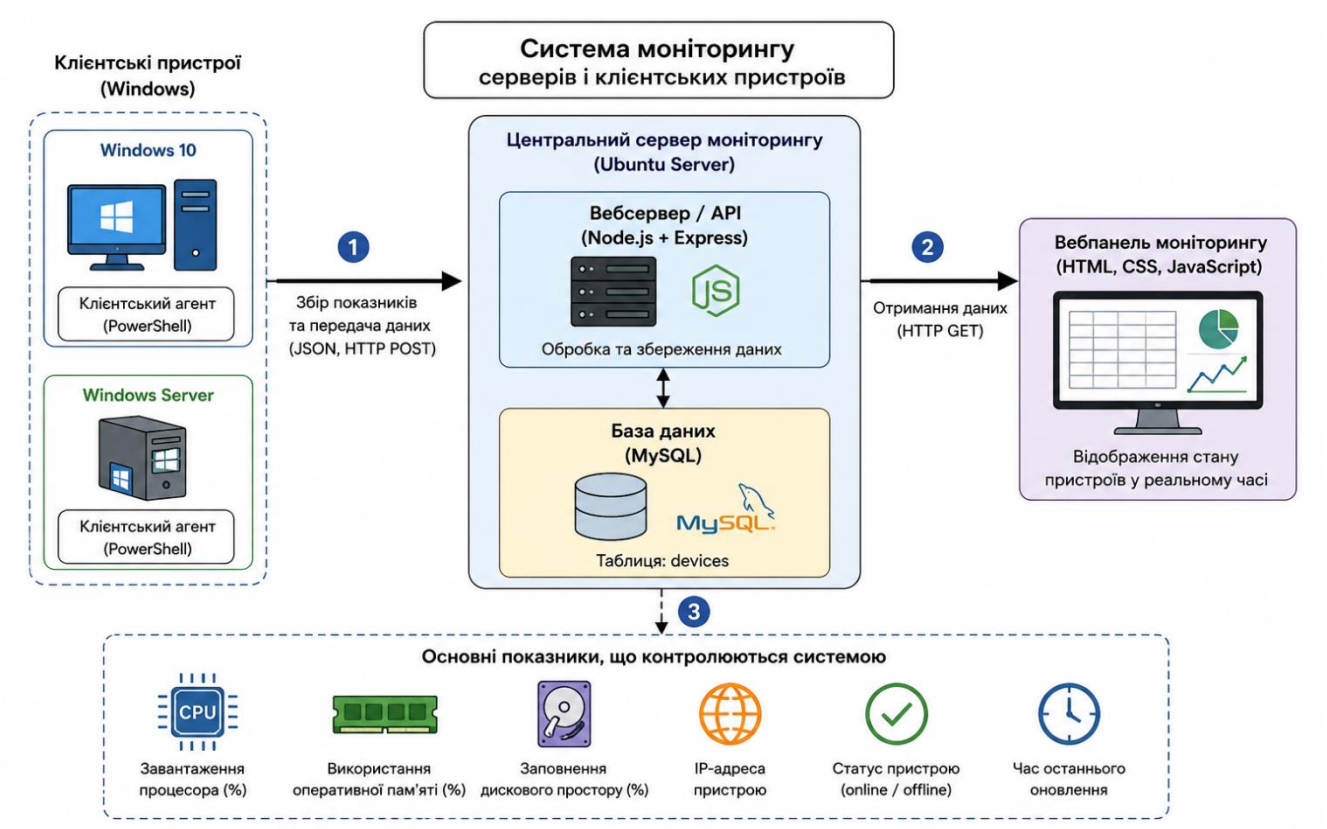

Рисунок 2.2 — Клієнт-серверна модель системи моніторингу

Джерело: розроблено автором

У межах дипломної роботи така модель є оптимальною, оскільки вона дозволяє поєднати теоретичні принципи побудови інформаційних систем із практичною реалізацією програмного стенду. Водночас вона не є надмірно складною, тому може бути реалізована у віртуальному середовищі на трьох машинах.

2.3 Вибір технологій для реалізації системи

Для реалізації системи моніторингу необхідно обрати набір технологій, які забезпечать роботу серверної частини, бази даних, клієнтського агента та вебінтерфейсу. У межах цієї роботи доцільно використати такі технології: Ubuntu Server, Node.js, Express, MySQL, PowerShell, HTML, CSS і JavaScript.

Ubuntu Server використовується як серверна операційна система. На ньому розміщується API, база даних і вебпанель. Вибір Linux-сервера є доцільним,

оскільки такі системи широко застосовуються для розгортання серверних застосунків, баз даних і вебслужб.

Node.js використовується для реалізації серверної частини. Згідно з офіційним описом Node.js, це безкоштовне, відкрите та кросплатформне середовище виконання JavaScript, яке дозволяє створювати сервери, вебзастосунки, інструменти командного рядка та скрипти [9]. Завдяки цьому Node.js підходить для створення API, яке прийматиме HTTP-запити від клієнтських агентів і повертатиме дані для вебпанелі.

Express використовується як вебфреймворк для Node.js. В офіційній документації Express зазначено, що він є легким і гнучким фреймворком маршрутизації з мінімальним набором базових можливостей, які можна розширювати за допомогою middleware [10]. Для цієї роботи Express є зручним, оскільки дозволяє швидко створити маршрути API, наприклад для приймання даних від клієнтів і для видачі списку пристроїв вебпанелі.

MySQL використовується як система керування базами даних. В офіційній документації MySQL зазначено, що MySQL є швидким, багатопотоковим, багатокористувацьким і надійним SQL-сервером баз даних [11]. У межах цієї роботи MySQL використовується для зберігання інформації про пристрої, їхні IP-адреси, показники CPU, RAM, Disk, статус і час останнього оновлення.

PowerShell використовується для реалізації клієнтського агента на Windows-пристроях. Згідно з документацією Microsoft, PowerShell є кросплатформним рішенням для автоматизації задач, яке включає командну оболонку, мову сценаріїв і середовище керування конфігурацією [12]. У цій системі PowerShell-скрипт збирає показники з Windows-пристрою та передає їх на сервер за допомогою HTTP-запиту.

HTML, CSS і JavaScript використовуються для створення вебпанелі. HTML визначає структуру сторінки, CSS відповідає за зовнішнє оформлення, а JavaScript забезпечує динамічне оновлення даних без повного перезавантаження сторінки. Завдяки цьому вебпанель може відображати актуальні показники пристроїв у зручному для адміністратора вигляді.

Таблиця 2.1 — Обґрунтування вибору технологій системи

Компонент системи	Обрана технологія	Призначення
Серверна операційна система	Ubuntu Server	Розміщення API, бази даних і вебпанелі
Серверне середовище	Node.js	Реалізація серверної логіки
Вебфреймворк	Express	Створення маршрутів API
База даних	MySQL	Збереження інформації про пристрої
Клієнтський агент	PowerShell	Збір показників із Windows-пристроїв
Вебінтерфейс	HTML, CSS, JavaScript	Відображення даних у браузері
Середовище тестування	VirtualBox	Розгортання віртуального стенду

Джерело: розроблено автором

Обрані технології забезпечують достатню функціональність для реалізації навчального стенду системи моніторингу. Вони є поширеними, мають доступну документацію та дозволяють реалізувати всі необхідні компоненти без використання складних корпоративних платформ.

2.4 Проєктування бази даних

База даних є одним із ключових компонентів системи моніторингу, оскільки саме вона забезпечує збереження інформації про пристрої та їхній поточний стан. У межах цієї роботи база даних повинна зберігати дані, які надходять від клієнтських агентів, і надавати їх серверній частині для відображення у вебпанелі.

З урахуванням функціональних вимог системи доцільно створити базу даних monitor_db, у якій буде таблиця devices. У наданих матеріалах запропоновано структуру таблиці, що містить поля id, name, ip, cpu, ram, disk, status і last_update . Така структура відповідає базовим потребам системи, оскільки дозволяє зберігати ідентифікатор пристрою, його назву, мережеву адресу, показники ресурсів, статус доступності та час останнього оновлення.

Таблиця 2.2 — Структура таблиці devices

Поле	Тип даних	Призначення

id	INT AUTO_INCREMENT PRIMARY KEY	Унікальний ідентифікатор запису
name	VARCHAR(100)	Назва пристрою
ip	VARCHAR(50)	IP-адреса пристрою
cpu	INT	Завантаження процесора, %
ram	INT	Використання оперативної пам’яті, %
disk	INT	Заповнення дискового простору, %
status	VARCHAR(20)	Поточний статус пристрою
last_update	TIMESTAMP	Час останнього оновлення даних

Джерело: розроблено автором

Поле id використовується як первинний ключ таблиці. Воно забезпечує унікальність кожного запису та дозволяє однозначно ідентифікувати пристрій або запис у базі даних. Поле name містить назву пристрою, що дає змогу адміністратору розуміти, який саме комп’ютер відображається у вебпанелі.

Поле ip призначене для збереження IP-адреси пристрою. Це важливо для мережевої діагностики, оскільки адміністратор може швидко визначити адресу вузла, з якого надходять дані. Поля cpu, ram і disk зберігають основні показники використання ресурсів. Для спрощення реалізації ці значення можуть зберігатися у відсотках як цілі числа.

Поле status використовується для збереження поточного стану пристрою. Наприклад, пристрій може мати статус online, якщо він нещодавно передавав дані, або offline, якщо оновлення не надходили протягом визначеного проміжку часу. Поле last_update містить часову мітку останнього отримання даних від пристрою. Саме на основі цього поля можна визначати актуальність інформації та формувати статус доступності.

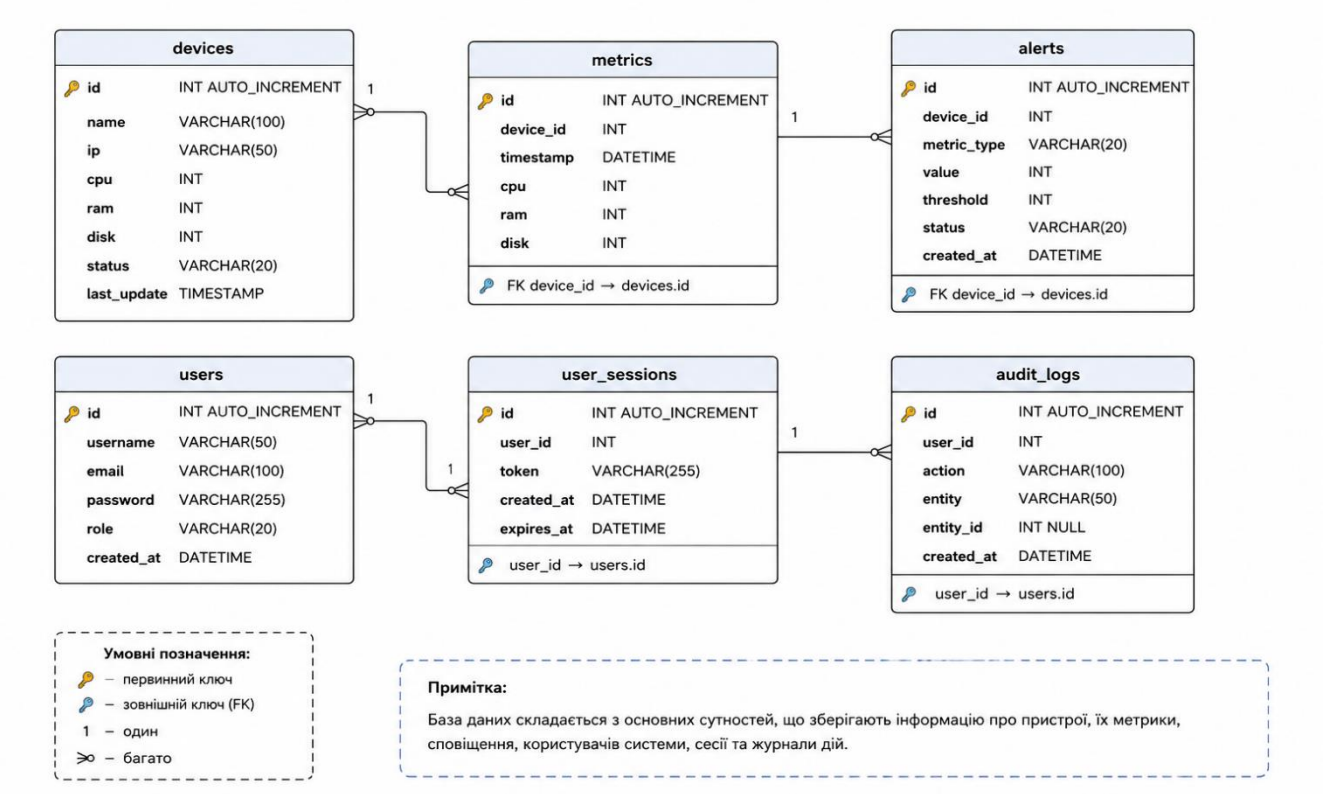

Рисунок 2.3 — Логічна структура бази даних системи моніторингу

Джерело: розроблено автором

У подальшому структура бази даних, що зображена на рис. 2.3 може бути розширена. Наприклад, можна додати окрему таблицю для історії метрик, щоб зберігати не лише останній стан пристрою, а й усі попередні значення. Це дозволило б будувати графіки зміни показників CPU, RAM і Disk за певний проміжок часу. Також можна додати таблицю користувачів для авторизації у вебпанелі, таблицю подій для збереження аварійних станів або таблицю налаштувань для визначення порогових значень.

Однак для базової реалізації дипломного проєкту таблиці devices достатньо, оскільки вона забезпечує збереження всіх основних показників, які потрібні для демонстрації роботи системи моніторингу.

2.5 Проєктування серверного API

Серверне API (рис. 2.4) є проміжним компонентом між клієнтськими агентами, базою даних і вебпанеллю. Його основне призначення полягає у прийманні даних від клієнтських пристроїв, перевірці отриманої інформації, записі або оновленні даних у базі даних, а також наданні вебпанелі актуального списку пристроїв.

У розроблюваній системі API працює за принципом HTTP-запитів. HTTP є протоколом прикладного рівня, який використовується для обміну повідомленнями між клієнтом і сервером; клієнт надсилає запит, а сервер повертає відповідь [13]. Така модель добре підходить для системи моніторингу, оскільки клієнтський агент може періодично надсилати на сервер структуровані дані про стан пристрою, а вебпанель може запитувати актуальні дані для відображення.

Для передавання даних між клієнтом і сервером доцільно використовувати формат JSON. JSON є текстовим форматом подання структурованих даних, який часто застосовується у вебзастосунках для передавання інформації між сервером і клієнтом [14]. У межах цієї роботи JSON-запит може містити назву пристрою, IP-адресу, завантаження CPU, використання RAM, заповнення Disk і статус пристрою.

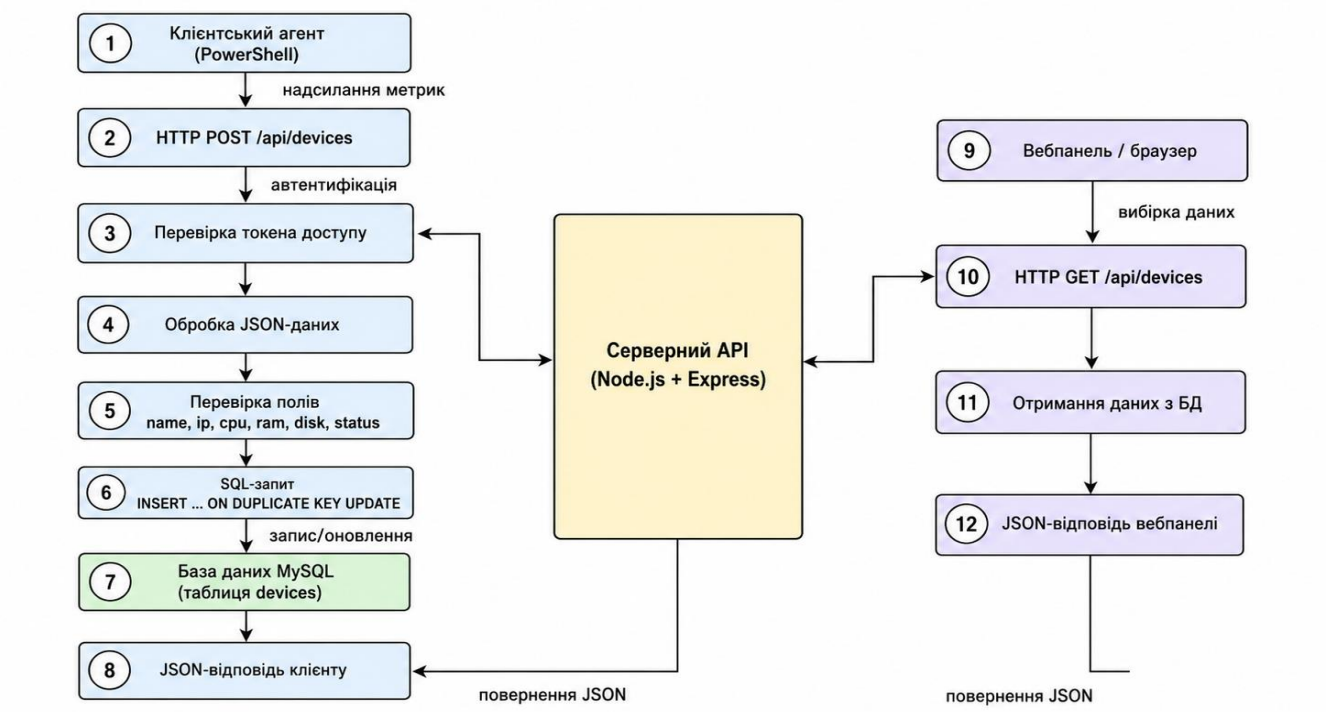

Рисунок 2.4 — Логіка роботи серверного API

Джерело: розроблено автором

Основним маршрутом API для приймання даних може бути маршрут:

POST /api/devices

Цей маршрут призначений для отримання показників від клієнтських агентів. Під час надходження запиту сервер повинен виконати декілька дій. Спочатку він перевіряє наявність і коректність токена авторизації, щоб запобігти надсиланню даних сторонніми пристроями. Далі сервер зчитує JSON-тіло запиту, перевіряє наявність потрібних полів і виконує запис або оновлення інформації в таблиці devices.

У наданих матеріалах практичної частини передбачено використання секретного токена ApiToken2026, який застосовується для обмеження доступу до API та запобігання несанкціонованому надсиланню метрик . Такий механізм не є повноцінною системою автентифікації корпоративного рівня, однак для навчального стенду він є достатнім, оскільки дозволяє показати базовий принцип захисту API.

Другим важливим маршрутом є:

GET /api/devices

Цей маршрут призначений для отримання списку пристроїв, які зберігаються в базі даних. Його використовує вебпанель для відображення актуального стану інфраструктури. Під час виконання такого запиту сервер звертається до бази даних, отримує записи з таблиці devices і повертає їх у форматі JSON.

Таблиця 2.3 — Основні маршрути серверного API

Метод	Маршрут	Призначення	Джерело запиту
POST	/api/devices	Передавання показників пристрою на сервер	PowerShell-агент
GET	/api/devices	Отримання списку пристроїв для відображення	Вебпанель
GET	/	Відкриття головної сторінки вебпанелі	Браузер адміністратора

Джерело: розроблено автором

Для реалізації API обрано Node.js і Express. Node.js забезпечує виконання серверного JavaScript-коду, а Express спрощує створення маршрутів, оброблення запитів і відповідей. З огляду на те, що система не потребує складної бізнес-логіки, така комбінація є достатньою для реалізації серверної частини.

Особливу увагу під час проєктування API необхідно приділити обробленню помилок. Якщо клієнтський агент надсилає некоректний токен, сервер повинен повернути відповідь із забороною доступу. Якщо у запиті відсутні потрібні дані, сервер має повідомити про помилку формату запиту. Якщо запис у базу даних не вдався, сервер повинен повернути повідомлення про внутрішню помилку. У документації MDN зазначено, що HTTP-коди стану показують, чи було успішно завершено запит, і поділяються на групи: інформаційні, успішні, перенаправлення, помилки клієнта та помилки сервера [15]. Це дозволяє коректно відображати результат оброблення запиту.

Отже, серверне API є центральним програмним компонентом системи. Воно забезпечує зв’язок між клієнтськими пристроями, базою даних і вебпанеллю, а також реалізує базову логіку контролю доступу та оброблення даних.

2.6 Проєктування клієнтського агента для Windows

Клієнтський агент є програмним компонентом, який виконується на Windows-пристрої та відповідає за збір показників стану системи. У межах цієї роботи агент реалізується у вигляді PowerShell-скрипта monitor.ps1. Такий варіант є доцільним, оскільки PowerShell входить до складу сучасних Windows-систем і дозволяє отримувати інформацію про процесор, оперативну пам’ять, диски та мережеві параметри без встановлення складного додаткового програмного забезпечення.

Основне завдання клієнтського агента (рис 2.5) полягає у періодичному збиранні таких показників:

- назва пристрою;
- IP-адреса;
- завантаження центрального процесора;
- використання оперативної пам’яті;
- заповнення дискового простору;
- статус пристрою;
- час передавання даних.

Після збирання цих даних агент формує JSON-об’єкт і надсилає його на серверне API за допомогою HTTP POST-запиту. У наданих матеріалах практичної частини наведено приклад використання PowerShell для отримання середнього завантаження процесора за допомогою команди Get-WmiObject win32_processor, а також передавання даних через Invoke-RestMethod .

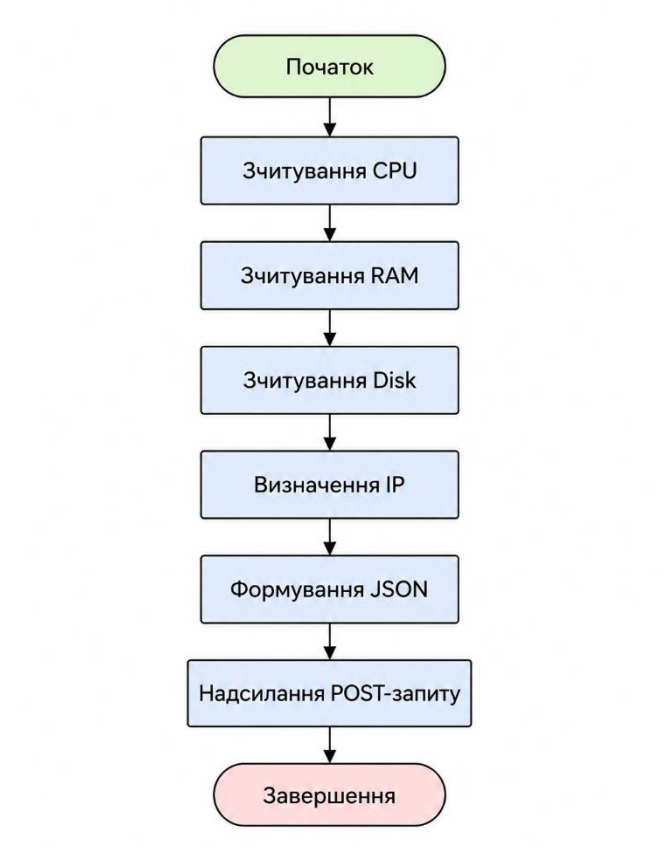

Рисунок 2.5 — Алгоритм роботи клієнтського агента

Джерело: розроблено автором

Клієнтський агент повинен знати адресу серверного API. У наданій інструкції з розгортання стенду зазначено, що після імпорту віртуальних машин і визначення IP-адреси Ubuntu Server потрібно відкрити файл monitor.ps1 на кожній Windows-машині та замінити стару IP-адресу в рядку $apiUrl = "http://192.168.1.110:3000/api/devices" на нову адресу сервера . Це є важливим етапом, оскільки без правильної IP-адреси клієнти не зможуть передавати дані на сервер.

Для автоматизації роботи агента можна використати Task Scheduler, тобто планувальник завдань Windows. У наданій інструкції зазначено, що клієнти автоматично починають відправляти дані за новою адресою, а завдання в Task Scheduler запускається щохвилини . Такий підхід дозволяє не запускати скрипт вручну кожного разу, а забезпечити регулярне оновлення даних.

Таблиця 2.4 — Дані, які передає клієнтський агент

Поле	Зміст	Приклад значення
name	Назва пристрою	WIN10-CLIENT
ip	IP-адреса пристрою	192.168.1.105
cpu	Завантаження процесора	35
ram	Використання оперативної пам’яті	62
disk	Заповнення диска	48
status	Стан пристрою	online

Джерело: розроблено автором

Важливою перевагою PowerShell-агента є простота редагування. Адміністратор може швидко змінити адресу сервера, токен авторизації, періодичність запуску або набір показників, які збираються. Крім того, такий агент легко продемонструвати під час захисту дипломної роботи, оскільки його логіка є зрозумілою та не потребує складного середовища розробки.

Обмеженням такого підходу є залежність від Windows-середовища. PowerShell-агент, розроблений для Windows, не може без змін використовуватися на Linux-пристроях. Однак архітектура системи дозволяє в майбутньому створити

окремий Linux-агент, який передаватиме дані у тому самому JSON-форматі на той самий API-маршрут.

2.7 Проєктування вебпанелі моніторингу

Вебпанель є інтерфейсом користувача, через який адміністратор переглядає інформацію про стан пристроїв. Вона повинна бути простою, зрозумілою та інформативною. Основним завданням вебпанелі є відображення списку пристроїв і їхніх поточних показників у вигляді таблиці або карток.

У межах цієї дипломної роботи вебпанель повинна відображати такі дані:

- назву пристрою;
- IP-адресу;
- завантаження CPU;
- використання RAM;
- заповнення Disk;
- онлайн/офлайн-статус;
- час останнього оновлення.

У наданому описі проєкту зазначено, що вебпанель (рис 2.6) повинна мати список пристроїв, статус і оновлення кожні 5 секунд, а дані можуть отримуватися за допомогою JavaScript-запиту fetch('/api/devices') . Такий підхід дозволяє оновлювати інформацію у браузері без повного перезавантаження сторінки.

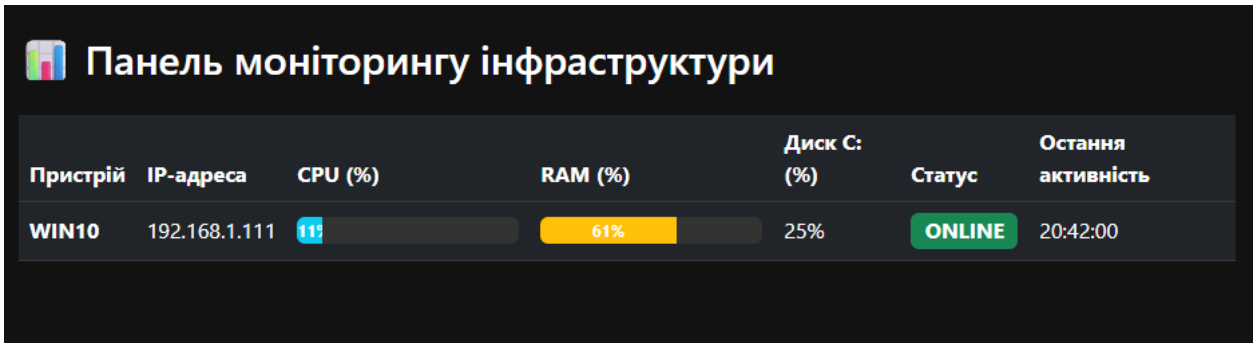

Рисунок 2.6 — Макет вебпанелі моніторингу

Джерело: розроблено автором

Для побудови інтерфейсу вебпанелі достатньо використати HTML, CSS і JavaScript. HTML визначає структуру сторінки, CSS забезпечує оформлення, а JavaScript виконує запити до API та оновлює таблицю. Такий варіант є зручним для дипломного проєкту, оскільки не потребує складних frontend-фреймворків і водночас дозволяє показати роботу динамічного вебінтерфейсу.

Для реалізації графіків у майбутньому можна використати бібліотеку Chart.js. В офіційній документації Chart.js зазначено, що бібліотека надає набір поширених типів графіків, плагінів і можливостей налаштування [16]. У межах базової версії системи графіки можуть бути додатковим функціоналом (рис. 2.7), однак їх наявність покращує наочність моніторингу й дозволяє відстежувати зміну показників у часі.

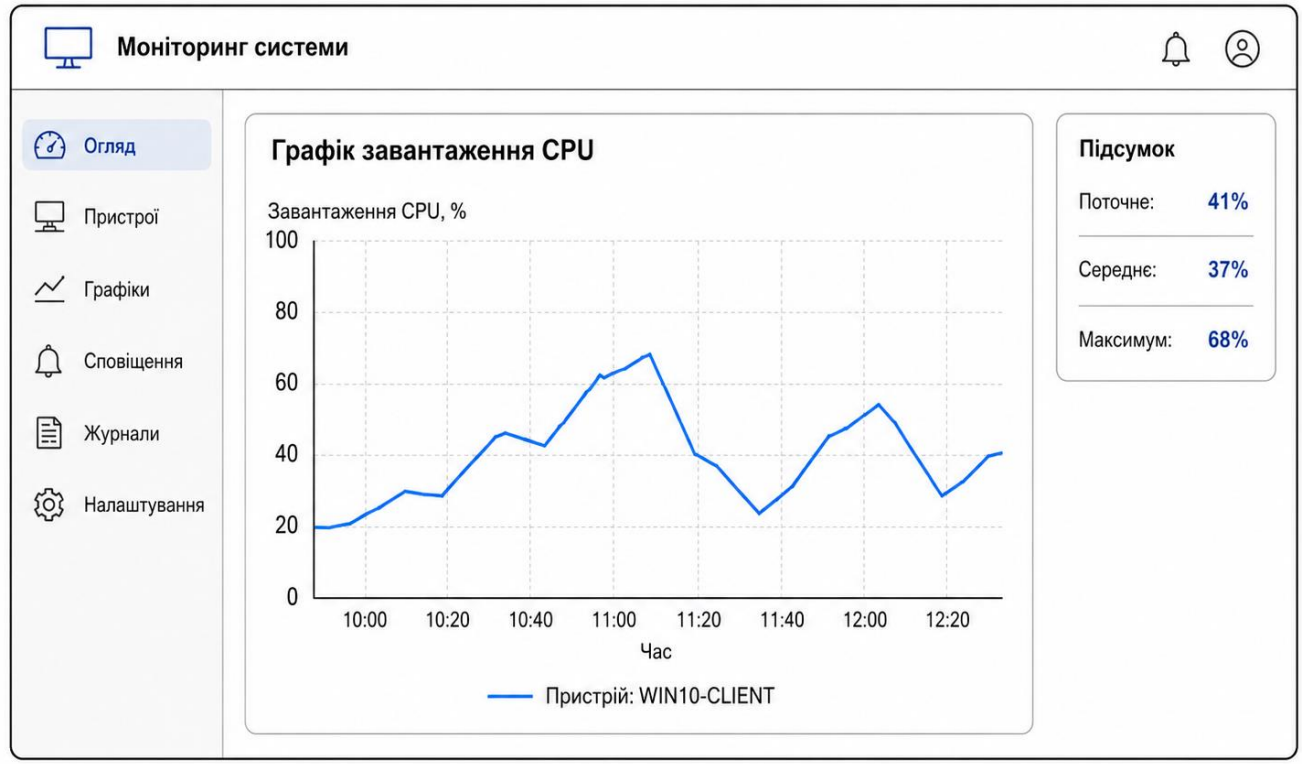

Рисунок 2.7 — Приклад відображення графіка завантаження CPU

Джерело: розроблено автором

Цей рисунок можна додати, якщо у практичній частині реалізовано або заплановано графіки. Якщо графіків немає, у тексті можна зазначити, що їх реалізація є напрямом подальшого розвитку системи.

Вебпанель також повинна відображати статус пристрою. Найпростішим способом визначення статусу є аналіз поля last_update. Якщо пристрій передавав дані протягом останнього заданого проміжку часу, наприклад 1–2 хвилин, він вважається активним. Якщо оновлення не надходили довше, пристрій може позначатися як offline. Такий підхід не потребує складних механізмів перевірки доступності, але дозволяє швидко оцінити актуальність даних.

Для зручності користувача доцільно передбачити візуальне виділення проблемних показників. Наприклад, якщо завантаження CPU або RAM перевищує певний поріг, відповідне значення може підсвічуватися. Це дозволить адміністратору швидше звернути увагу на потенційно проблемний пристрій.

Отже, вебпанель є завершальним елементом системи моніторингу, який перетворює технічні дані з бази даних у зрозумілу для користувача форму. Вона забезпечує оперативний перегляд стану пристроїв і робить систему придатною для практичного використання.

2.8 Проєктування віртуального середовища для розгортання системи

Для тестування системи моніторингу доцільно використовувати віртуальне середовище. Це дозволяє створити кілька ізольованих машин на одному фізичному комп’ютері та змоделювати роботу локальної мережі без використання окремого фізичного обладнання.

У межах дипломної роботи віртуальний стенд складається з трьох машин:

- Ubuntu Server — сервер API, бази даних і вебпанелі;
- Windows 10 — клієнтський пристрій;
- Windows Server — другий клієнтський пристрій.

Для створення лабораторного середовища потрібно встановити VirtualBox, Ubuntu Server, Windows Server 2019/2022, Windows 10, Visual Studio Code, Postman

і, за потреби, Git . Таке середовище (рис. 2.8) дозволяє повністю відтворити роботу системи моніторингу: серверна машина приймає дані, а дві клієнтські машини передають свої показники.

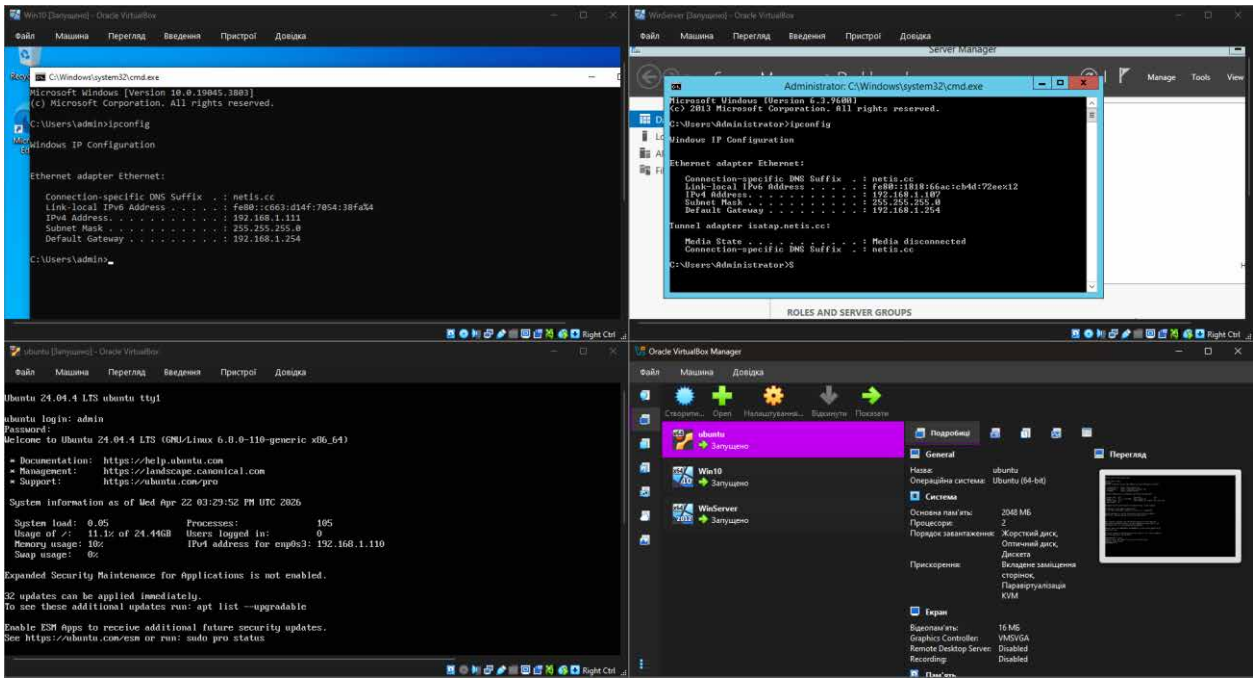

Рисунок 2.8 — Схема віртуального стенду системи моніторингу

Джерело: розроблено автором

Мережеве з’єднання між віртуальними машинами має бути налаштоване таким чином, щоб усі вони могли бачити одна одну. У наданій інструкції з розгортання стенду зазначено, що машини використовують режим **Bridged Adapter**, тобто мережевий міст, і його потрібно прив’язати до актуальної фізичної мережевої карти комп’ютера — Wi-Fi або Ethernet . Це дозволяє віртуальним машинам отримати IP-адреси з тієї самої мережі, що й фізичний комп’ютер.

Після запуску Ubuntu Server необхідно визначити його IP-адресу за допомогою команди ip a. Саме ця адреса використовується у веббраузері для відкриття панелі моніторингу та в клієнтських скриптах monitor.ps1 для передавання даних на API .

Використання віртуального середовища має кілька переваг. По-перше, воно дозволяє швидко створити ізольований стенд для демонстрації роботи системи. По-друге, у разі помилки налаштування можна відновити або перевстановити окрему віртуальну машину без впливу на основну операційну систему. По-третє, такий

підхід зручний для захисту дипломної роботи, оскільки всі компоненти системи можна продемонструвати на одному комп’ютері.

Отже, віртуальне середовище є важливою частиною практичної реалізації дипломного проєкту. Воно дозволяє перевірити працездатність системи моніторингу, протестувати взаємодію клієнтів із сервером і показати роботу вебпанелі в умовах, наближених до реальної локальної мережі.

2.9 Висновки до розділу 2

У другому розділі було спроєктовано систему моніторингу серверів і пристроїв. Визначено, що найдоцільнішою для цієї роботи є клієнт-серверна архітектура, у якій Windows-пристрої виконують роль клієнтів, Ubuntu Server — роль центрального сервера, MySQL — роль сховища даних, а вебпанель — роль інтерфейсу адміністратора.

Обґрунтовано вибір технологій для реалізації системи. Для серверної частини обрано Node.js і Express, для зберігання даних — MySQL, для клієнтського агента — PowerShell, а для вебінтерфейсу — HTML, CSS і JavaScript. Такий набір технологій є достатнім для реалізації навчального стенду та дозволяє продемонструвати всі ключові етапи роботи системи моніторингу.

Спроєктовано структуру бази даних monitor_db і таблиці devices, яка зберігає назву пристрою, IP-адресу, показники CPU, RAM, Disk, статус і час останнього оновлення. Також визначено основні маршрути серверного API, логіку роботи клієнтського агента та структуру вебпанелі.

Окремо розглянуто віртуальне середовище для розгортання системи. Встановлено, що використання трьох віртуальних машин — Ubuntu Server, Windows 10 і Windows Server — дозволяє змоделювати роботу локальної інфраструктури та протестувати передавання даних від клієнтів до сервера.

Таким чином, у другому розділі сформовано архітектурну та технологічну основу для подальшої практичної реалізації системи моніторингу.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ВЕБПАНЕЛІ ТА ПРОГРАМНОГО СТЕНДУ СИСТЕМИ МОНІТОРИНГУ

3.1 Створення віртуального лабораторного середовища

Практична реалізація системи моніторингу серверів і пристроїв виконувалася у віртуальному лабораторному середовищі. Використання віртуалізації дає змогу створити декілька ізольованих операційних систем на одному фізичному комп’ютері та змоделювати роботу локальної мережі без потреби у додатковому фізичному обладнанні. У межах дипломної роботи такий підхід є доцільним, оскільки дозволяє розгорнути серверну частину, клієнтські пристрої та перевірити їхню взаємодію в контрольованому середовищі.

Для створення лабораторного стенду було використано Oracle VM VirtualBox. У системі створено три віртуальні машини (рис. 3.1): **Ubuntu Server**, **Windows 10** та **Windows Server**. Ubuntu Server виконує роль центрального сервера, на якому розміщуються API, база даних і вебпанель. Windows 10 і Windows Server виконують роль клієнтських пристроїв, які збирають локальні показники стану системи та передають їх на сервер. Така структура відповідає практичній логіці проєкту, у якій передбачено сервер збору даних, клієнти та вебпанель для відображення результатів моніторингу .

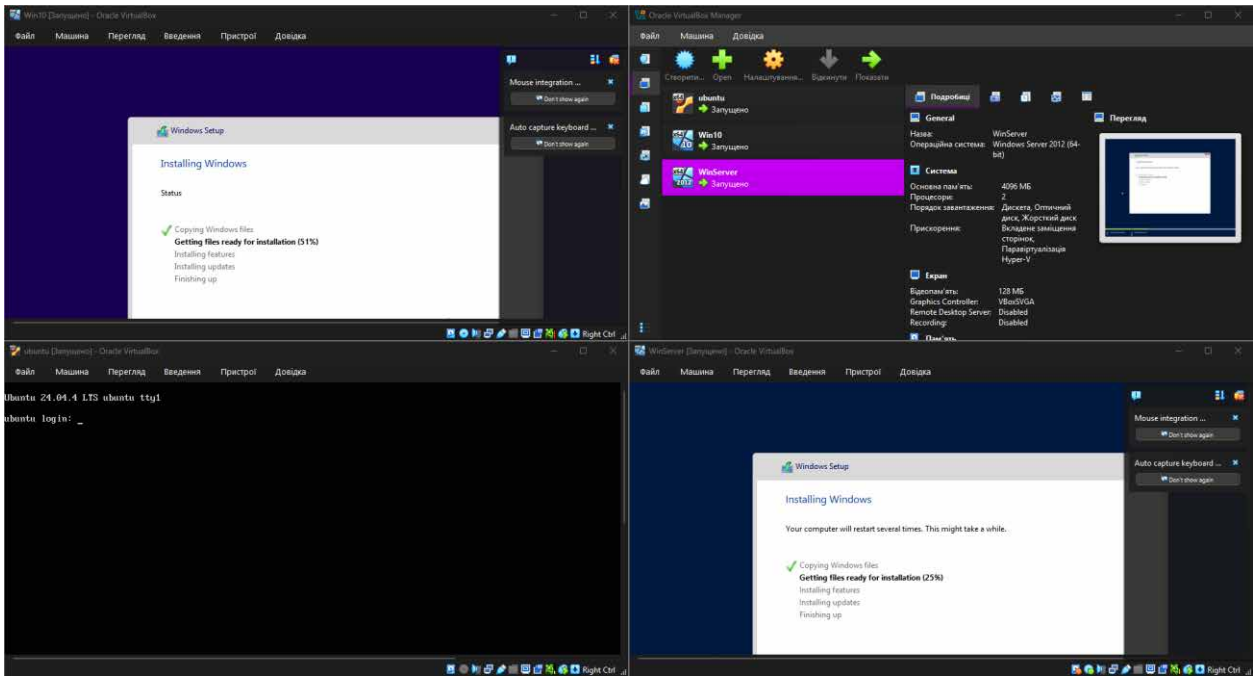

Рисунок 3.1 — Створення трьох віртуальних машин у VirtualBox

Джерело: розроблено автором

Для забезпечення мережевої взаємодії між віртуальними машинами було використано режим мережевого мосту **Bridged Adapter**. У документації Oracle VirtualBox зазначено, що для ввімкнення bridged networking необхідно відкрити налаштування віртуальної машини, перейти на сторінку **Network**, у полі **Attached To** вибрати **Bridged Network**, а також обрати фізичний мережевий інтерфейс хостової системи [17]. Це дозволяє віртуальній машині працювати в локальній мережі як окремому вузлу, що є важливим для взаємодії Windows-клієнтів із сервером Ubuntu.

У наданій інструкції з розгортання стенду також підкреслено, що після імпорту віртуальних машин необхідно для кожної з них налаштувати мережу в режимі **Мережевий міст** і прив’язати її до поточної фізичної мережевої карти комп’ютера — Wi-Fi або Ethernet . Це налаштування є критично важливим, оскільки саме воно забезпечує отримання IP-адрес із локальної мережі та можливість передавання даних між клієнтами й сервером.

Після налаштування мережі всі віртуальні машини повинні отримати IP-адреси в одній підмережі. У матеріалах практичної реалізації зазначено, що всі три машини знаходяться в одній підмережі типу 192.168.1.X і використовують один шлюз, що підтверджує коректну роботу мережевого мосту . Наявність спільної підмережі є необхідною умовою, щоб клієнтські агенти могли надсилати HTTP-запити на серверне API.

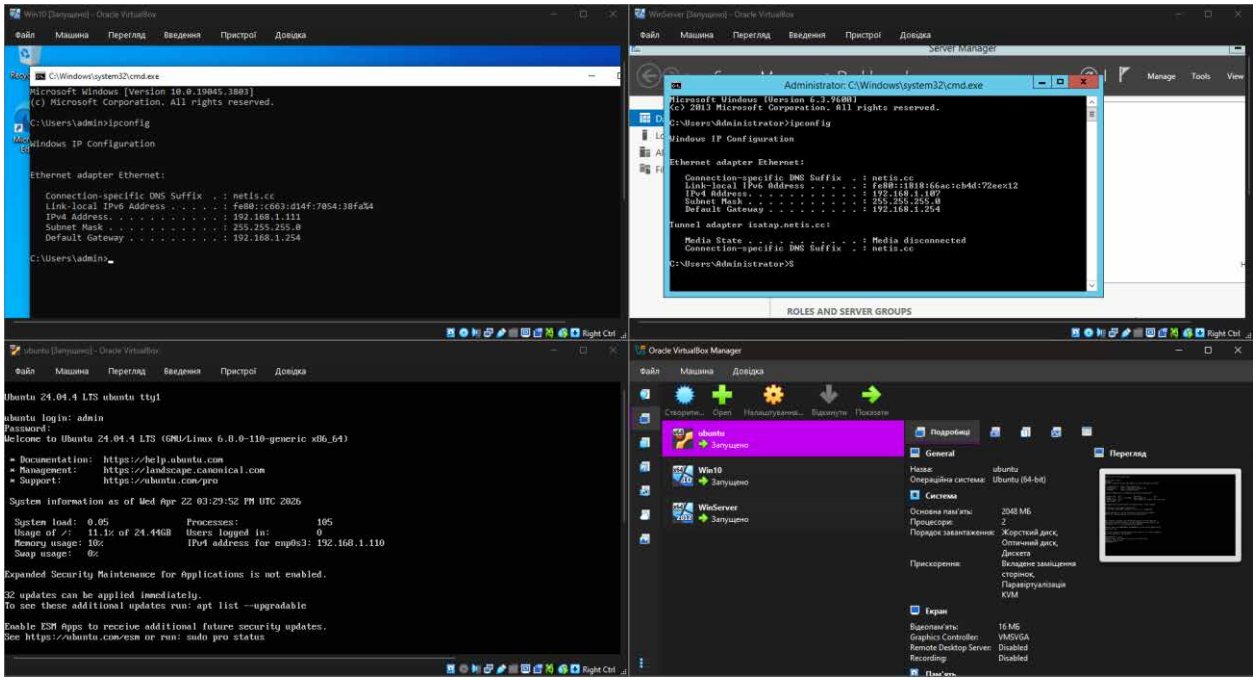

Рисунок 3.2 — Перевірка IP-адрес віртуальних машин у локальній мережі

Джерело: розроблено автором

Для визначення IP-адреси Ubuntu Server використовується команда:

ip a

Отримана IP-адреса надалі використовується у двох місцях: у браузері адміністратора для відкриття вебпанелі та в клієнтських PowerShell-скриптах для передавання даних на API. В інструкції з розгортання стенду зазначено, що після запуску Ubuntu Server потрібно виконати команду ip a, знайти нову IP-адресу сервера та записати її для подальшого використання .

Таким чином, на першому етапі було підготовлено віртуальне середовище, яке імітує невелику локальну IT-інфраструктуру. У ньому Ubuntu Server виконує роль

центрального вузла моніторингу, а Windows 10 і Windows Server — роль клієнтських пристроїв, що передають показники свого стану.

3.2 Налаштування Ubuntu Server

Ubuntu Server є центральним компонентом розробленої системи моніторингу. На ньому розгортаються серверне API, база даних MySQL і статичні файли вебпанелі. Вибір серверної операційної системи на базі Linux є доцільним, оскільки такі системи широко використовуються для розміщення вебсервісів, баз даних та прикладного серверного програмного забезпечення.

Підготовка серверного середовища починається з оновлення списку пакетів. В офіційній документації Ubuntu Server зазначено, що керування програмним забезпеченням в Ubuntu виконується через Debian-пакети, Snap-пакети, APT-команди та репозиторії [18]. У межах роботи APT використовується для встановлення необхідних компонентів серверної частини (рис. 3.3).

Для оновлення списку пакетів використовується команда:

sudo apt update

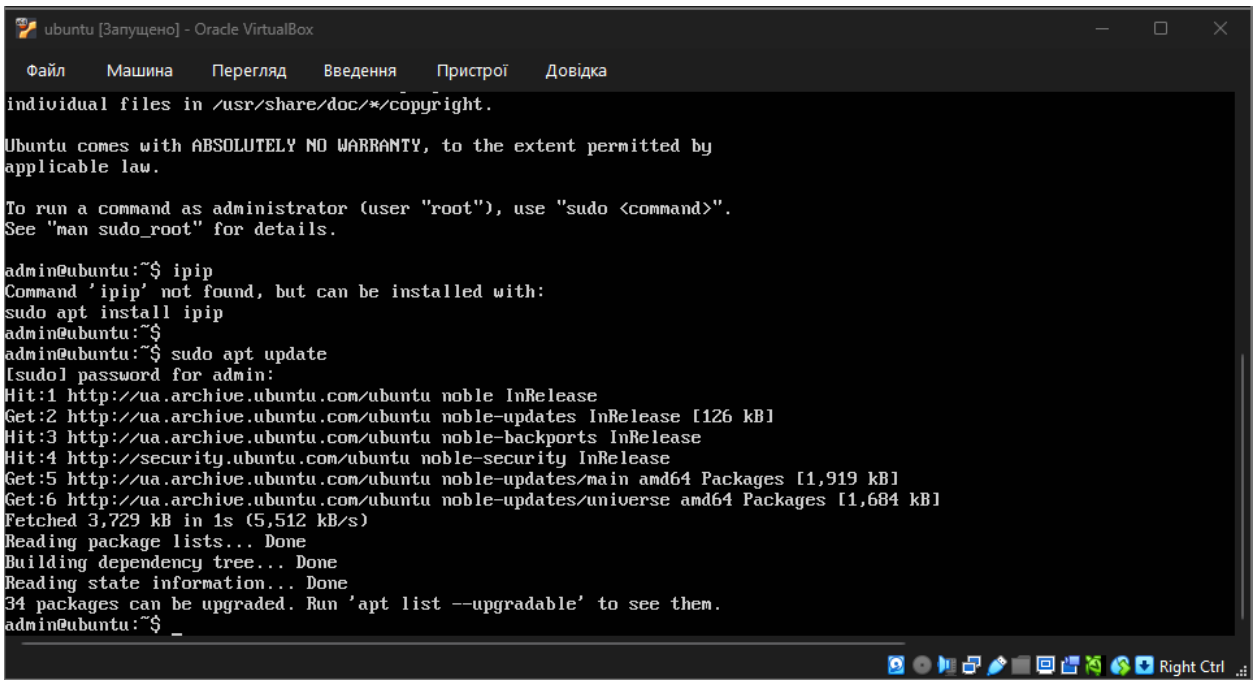

Рисунок 3.3 — Оновлення списку пакетів Ubuntu Server

Джерело: розроблено автором

Після оновлення списку пакетів встановлюються Node.js, npm і MySQL Server (рис. 3.4). У практичних матеріалах проєкту зазначено, що для серверної частини може використовуватися Node.js, а встановлення виконується командою sudo apt install nodejs npm . Node.js потрібен для запуску серверного API, а npm використовується для встановлення необхідних пакетів JavaScript-проєкту.

sudo apt install nodejs npm

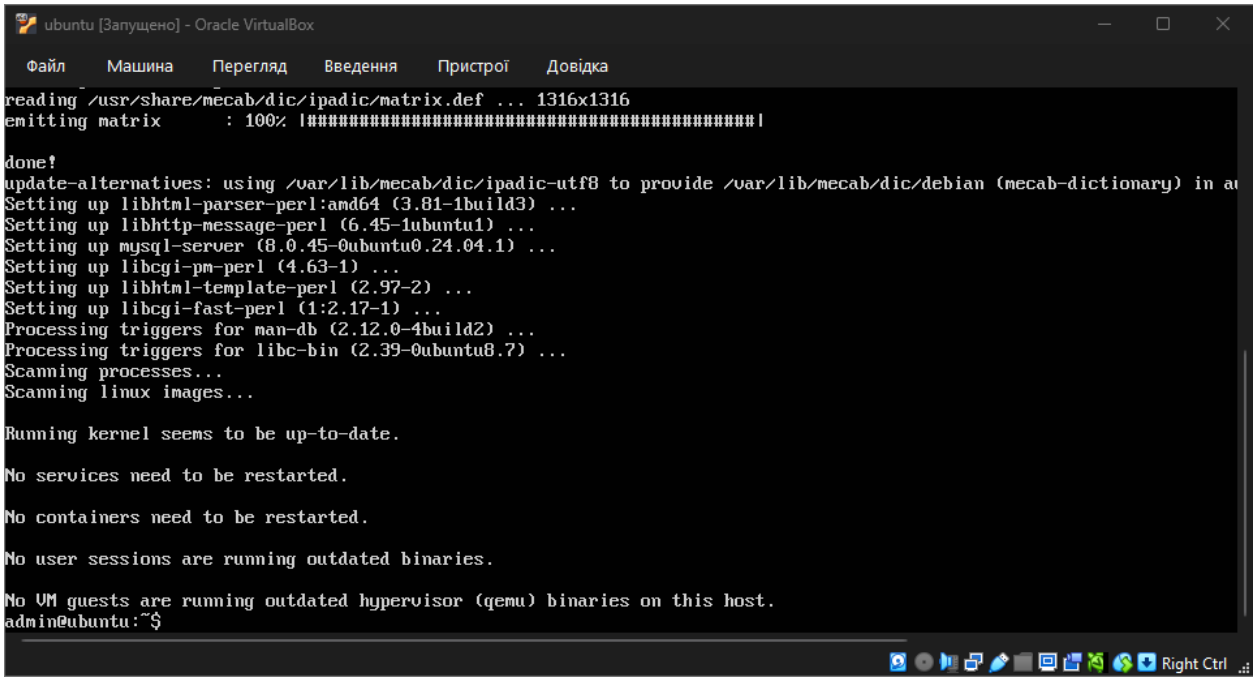

Рисунок 3.4 — Встановлення Node.js та npm на Ubuntu Server

Джерело: розроблено автором

Для збереження даних системи моніторингу використовується MySQL Server. У документації MySQL зазначено, що робота з базою даних передбачає створення бази, створення таблиць, завантаження даних і виконання запитів для їх отримання [19]. У межах дипломного проєкту MySQL використовується для збереження даних про пристрої, їхні IP-адреси, показники CPU, RAM, Disk, статус і час останнього оновлення.

sudo apt install mysql-server

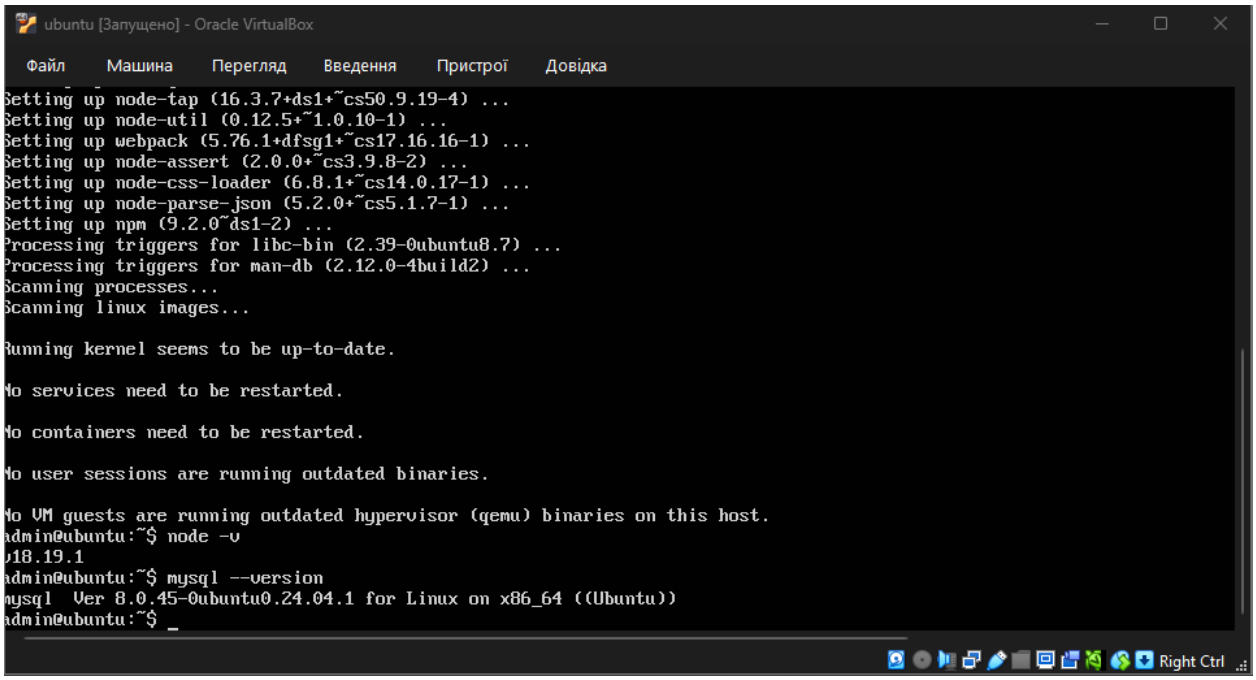

Рисунок 3.5 — Встановлення MySQL Server на Ubuntu Server

Джерело: розроблено автором

Після встановлення MySQL (рис. 3.5) виконується вхід у консоль керування базою даних:

sudo mysql

Далі створюється база даних monitor_db (рис. 3.6):

CREATE DATABASE monitor_db;

USE monitor_db;

У практичних матеріалах також зазначено, що після встановлення Node.js, npm і MySQL виконується вхід у консоль MySQL, створення бази даних і перехід до неї .

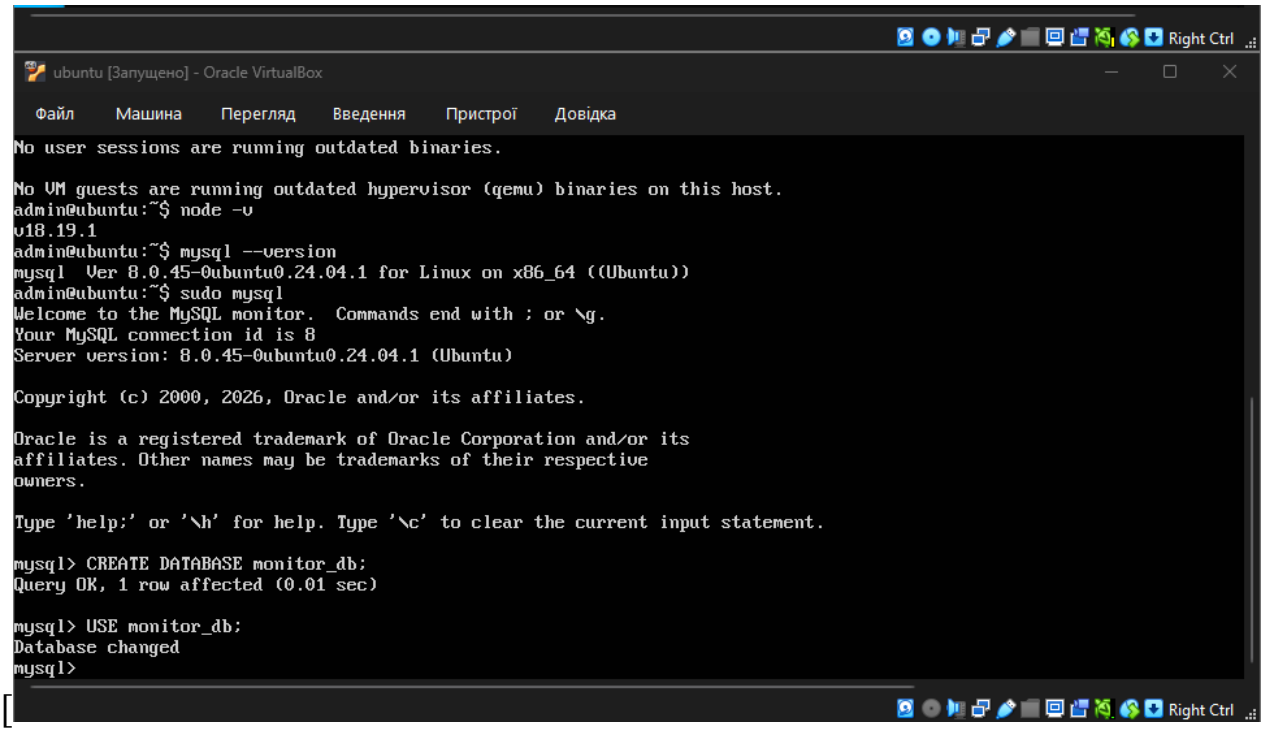

Рисунок 3.6 — Створення бази даних monitor_db

Джерело: розроблено автором

Для підключення API до бази даних доцільно використовувати окремий обліковий запис, а не користувача root. Такий підхід відповідає базовому принципу обмеження прав доступу, коли застосунок отримує лише ті права, які потрібні для виконання його функцій. У Законі України «Про захист інформації в інформаційно-комунікаційних системах» визначено, що захист інформації в системі передбачає діяльність, спрямовану на запобігання несанкціонованим діям щодо інформації [20]. У межах цієї системи використання окремого користувача бази даних є одним із простих організаційно-технічних заходів зменшення ризику несанкціонованого доступу.

У практичній реалізації API підключається до бази даних через користувача api_user, пароль ApiPass2026! і базу monitor_db (рис. 3.7).

Приклад створення користувача може мати такий вигляд:

```
CREATE USER 'api_user'@'localhost' IDENTIFIED BY 'ApiPass2026!';
GRANT ALL PRIVILEGES ON monitor_db.* TO 'api_user'@'localhost';
FLUSH PRIVILEGES;
```

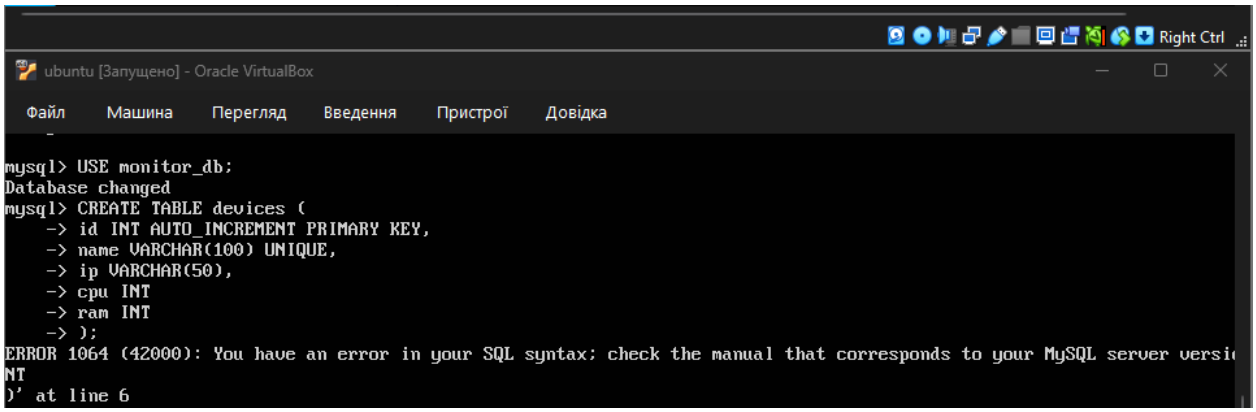

Рисунок 3.7 — Створення користувача api_user для підключення API до бази даних

Джерело: розроблено автором

Отже, на етапі налаштування Ubuntu Server було підготовлено всі необхідні серверні компоненти: оновлено систему, встановлено Node.js, npm і MySQL Server, створено базу даних monitor_db та підготовлено окремого користувача для підключення API.

3.3 Створення бази даних і таблиці пристроїв

База даних monitor_db призначена для збереження інформації про пристрої, які підключаються до системи моніторингу. Основною таблицею є таблиця devices (рис. 3.8), у якій зберігаються актуальні параметри кожного клієнтського пристрою.

Структура таблиці розроблена з урахуванням функціональних вимог системи. У наданих матеріалах зазначено, що система повинна відображати CPU, RAM, Disk, IP-адреси, онлайн/офлайн-статус і час останньої активності . Тому таблиця повинна містити поля для ідентифікації пристрою, збереження його мережевої адреси, ресурсних показників і часової мітки оновлення.

SQL-запит для створення таблиці:

```
CREATE TABLE devices (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    ip VARCHAR(50),
    cpu INT,
    ram INT,
    disk INT,
    status VARCHAR(20),
    last_update TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

Рисунок 3.8 — Створення таблиці devices у MySQL

Джерело: розроблено автором

Поле id є первинним ключем таблиці. Воно автоматично збільшується при створенні нового запису та забезпечує унікальну ідентифікацію рядка. Поле name використовується для збереження назви пристрою, наприклад імені комп'ютера Windows. Поле ip містить IP-адресу клієнтської машини, що дозволяє адміністратору швидко визначити пристрій у мережі.

Поля cpu, ram і disk зберігають числові значення у відсотках. Вони використовуються для оцінювання поточного навантаження на пристрій. Поле status призначене для збереження стану пристрою, наприклад online. Поле

last_update автоматично оновлюється при зміні запису та використовується для визначення актуальності отриманих даних.

Таблиця 3.1 — Призначення полів таблиці devices

Поле	Тип даних	Призначення
id	INT AUTO_INCREMENT PRIMARY KEY	Унікальний ідентифікатор запису
name	VARCHAR(100)	Назва пристрою
ip	VARCHAR(50)	IP-адреса пристрою
cpu	INT	Завантаження процесора у відсотках
ram	INT	Використання оперативної пам’яті у відсотках
disk	INT	Заповнення дискового простору у відсотках
status	VARCHAR(20)	Поточний стан пристрою
last_update	TIMESTAMP	Час останнього оновлення даних

Джерело: розроблено автором

Для уникнення дублювання записів у таблиці доцільно використовувати механізм оновлення наявного запису пристрою замість створення нового рядка при кожному запиті. MySQL підтримує конструкцію INSERT ... ON DUPLICATE KEY UPDATE, яка дозволяє виконати оновлення запису, якщо під час вставлення виникає конфлікт унікального ключа [21]. У практичній реалізації такий підхід використовується для того, щоб у таблиці зберігався актуальний стан пристрою, а не велика кількість повторюваних записів.

Для коректної роботи такого механізму доцільно зробити поле name або ip унікальним. Наприклад:

ALTER TABLE devices ADD UNIQUE KEY unique_name (name);

Після цього серверне API може виконувати вставлення або оновлення даних в одному SQL-запиті. Це спрощує логіку роботи серверної частини та зменшує кількість надлишкових записів у базі даних.

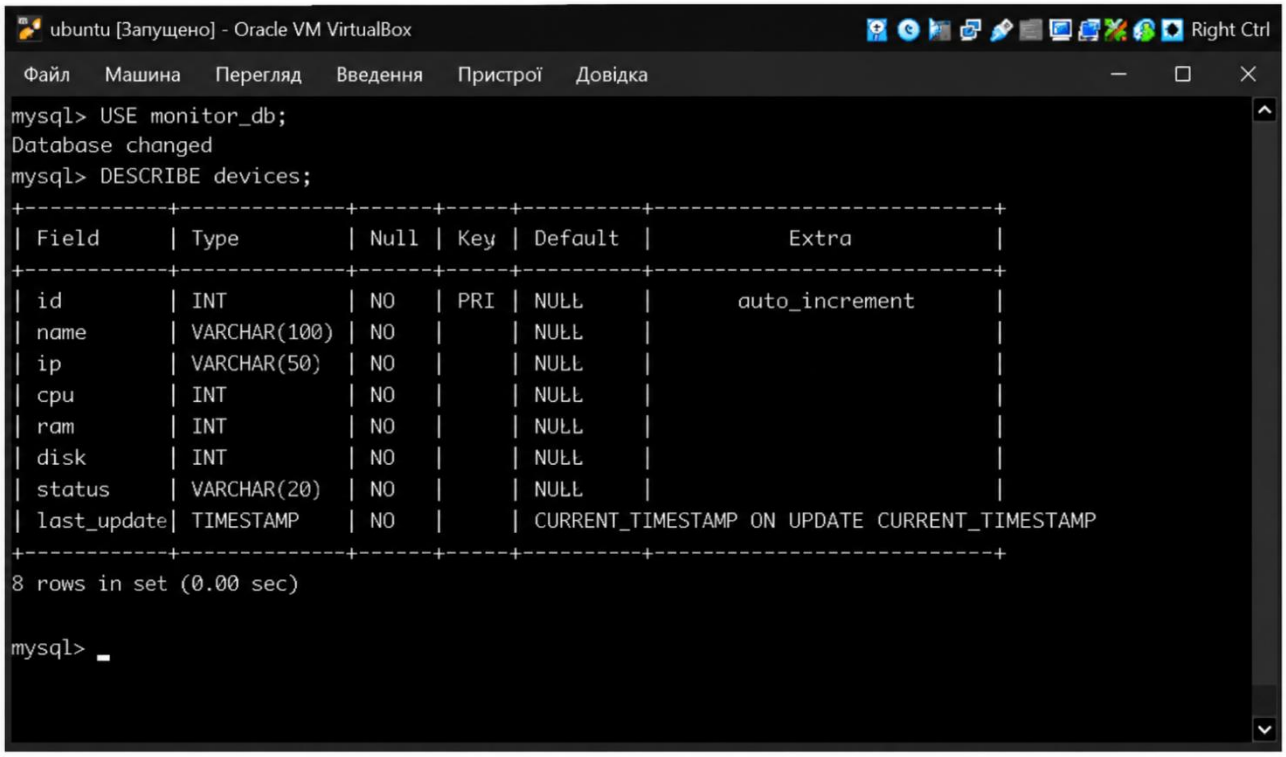

Рисунок 3.9 — Перегляд структури таблиці devices у MySQL

Джерело: розроблено автором

Отже, база даних monitor_db і таблиця devices (рис. 3.9) забезпечують централізоване збереження основних показників стану клієнтських пристроїв. Така структура є достатньою для базової реалізації системи моніторингу й водночас може бути розширена в майбутньому шляхом додавання таблиці історії метрик, користувачів або журналу подій.

3.4 Реалізація серверного API

Після підготовки бази даних було реалізовано серверне API на основі Node.js та Express. API є центральним програмним компонентом системи, оскільки саме воно приймає дані від клієнтських агентів, записує їх у MySQL і надає вебпанелі актуальну інформацію для відображення.

Node.js використовується як середовище виконання серверного JavaScript-коду, а Express — як фреймворк для створення HTTP-маршрутів. Як зазначалося раніше, Express є легким вебфреймворком для Node.js, який спрощує

маршрутизацію та оброблення запитів [10]. У межах цієї системи він використовується для створення маршрутів POST /api/devices і GET /api/devices.

У серверному коді використовуються бібліотеки:

- express — для створення вебсервера та маршрутів;
- mysql2 — для підключення до MySQL;
- body-parser — для оброблення JSON-тіла запиту;
- cors — для керування міждоменними HTTP-запитами.

Для встановлення залежностей (рис. 3.10) застосовується npm. npm є стандартним менеджером пакетів для екосистеми Node.js і використовується для встановлення бібліотек, необхідних для роботи застосунку [22]. У межах цієї роботи npm застосовується для підготовки серверного проєкту та встановлення модулів Express, MySQL2, body-parser і cors.

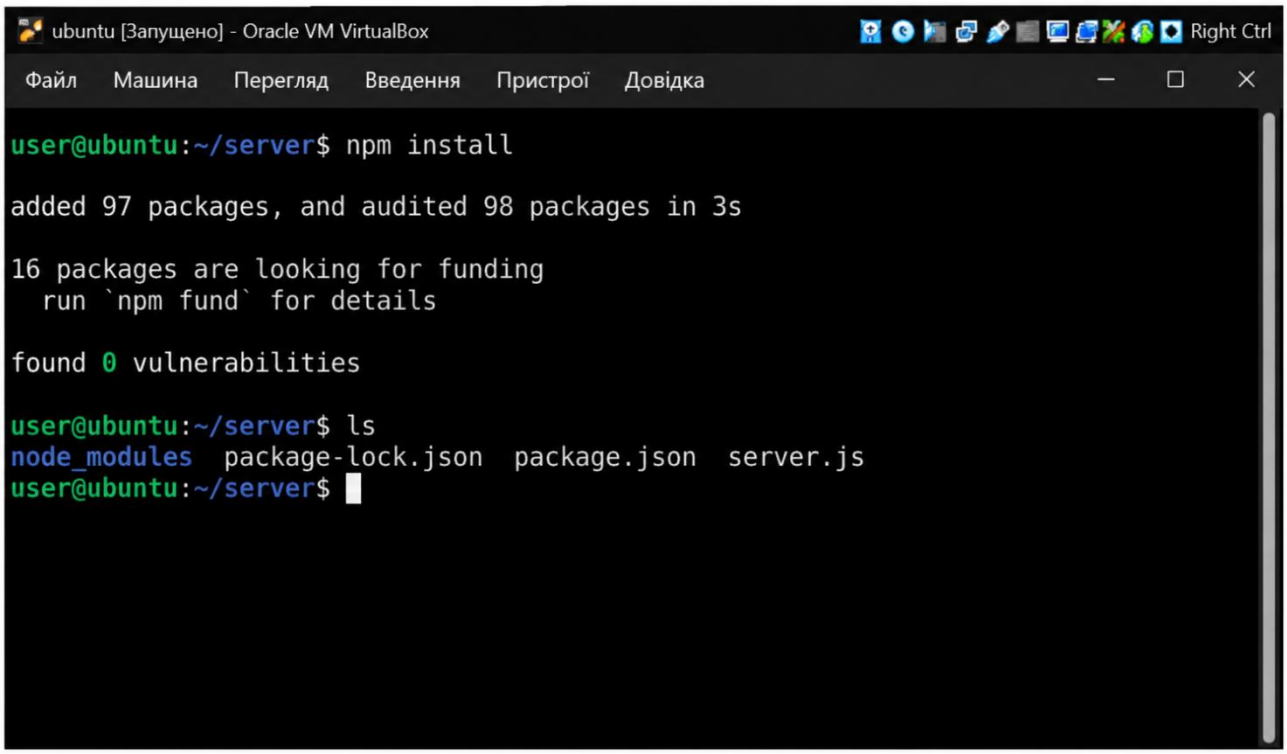

Рисунок 3.10 — Встановлення залежностей серверного проєкту

Джерело: розроблено автором

Основна структура серверного застосунку має такий вигляд (лістинг 3.1):

```
const express = require('express');
const mysql = require('mysql2');
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();

app.use(cors());
app.use(bodyParser.json());
app.use(express.static('public'));

const SECRET_TOKEN = 'ApiToken2026';

const db = mysql.createPool({
    host: '127.0.0.1',
    user: 'api_user',
    password: 'ApiPass2026!',
    database: 'monitor_db'
});
```

Лістинг 3.1 – Структура серверного застосунку

Джерело: розроблено автором

У практичних матеріалах наведено аналогічний фрагмент коду, де створюється Express-застосунок, підключаються cors, bodyParser.json(), статична папка public, секретний токен ApiToken2026 і пул підключення до бази monitor_db.

Рисунок 3.12 — Основна структура серверного застосунку Node.js

Для базового захисту API використовується перевірка токена доступу. Клієнтський агент повинен передавати заголовок Authorization зі значенням Bearer ApiToken2026. Якщо токен неправильний або відсутній, сервер не повинен приймати дані. Такий механізм є спрощеним варіантом контролю доступу, але він демонструє базовий принцип обмеження несанкціонованих запитів до API.

Питання захисту API є важливим, оскільки інтерфейси прикладного програмування часто стають точкою взаємодії між компонентами системи. OWASP API Security Top 10 визначає типові ризики безпеки API, серед яких порушення авторизації, помилки автентифікації та недостатній контроль доступу [23]. У межах дипломного стенду використання токена не є повноцінною корпоративною системою безпеки, однак воно дозволяє показати необхідність перевірки джерела запитів.

```
const authenticate = (req, res, next) => {
    const authHeader = req.headers['authorization'];

     if (authHeader && authHeader === `Bearer ${SECRET_TOKEN}`) {
        next();
    } else {
         res.status(403).json({ error: 'Access Denied: Invalid
 Token' });
    }
};
```

Лістинг 3.2 – Перевірка валідності токену доступу до API

Джерело: розроблено автором

Основний маршрут для приймання даних зображений у додатку А.

Цей маршрут приймає JSON-дані від клієнтського агента, перевіряє наявність назви пристрою та IP-адреси, після чого записує або оновлює інформацію в базі даних. Передавання даних у форматі JSON відповідає сучасній практиці

веброзробки, оскільки JSON є зручним текстовим форматом обміну структурованими даними між клієнтом і сервером [14].

Для вебпанелі реалізовано маршрут отримання списку пристроїв:

```
app.get('/api/devices', (req, res) => {
    db.query('SELECT * FROM devices', (err, results) => {
        if (err) {
            return res.status(500).json({ error: 'Database error' });
        }

        res.json(results);
    });
});
```

Лістинг 3.3 – Маршрут отримання списку пристроїв для вебпанелі

Джерело: розроблено автором

Цей маршрут виконує SQL-запит до таблиці devices і повертає результат у форматі JSON. Його використовує JavaScript-код вебпанелі для заповнення таблиці пристроїв.

Після опису маршрутів сервер запускається на порту 3000:

```
const PORT = 3000;

app.listen(PORT, () => {
    console.log(`API Server is running on port ${PORT}`);
});
```

Лістинг 3.4 – Ініціалізація порта для прослуховування

Джерело: розроблено автором

У наданій інструкції з розгортання стенду у додатку Б зазначено, що вебпанель відкривається у браузері за адресою http://<НОВА_IP_АДРЕСА_UBUNTU>:3000, що відповідає використанню порту 3000 для роботи серверного застосунку .

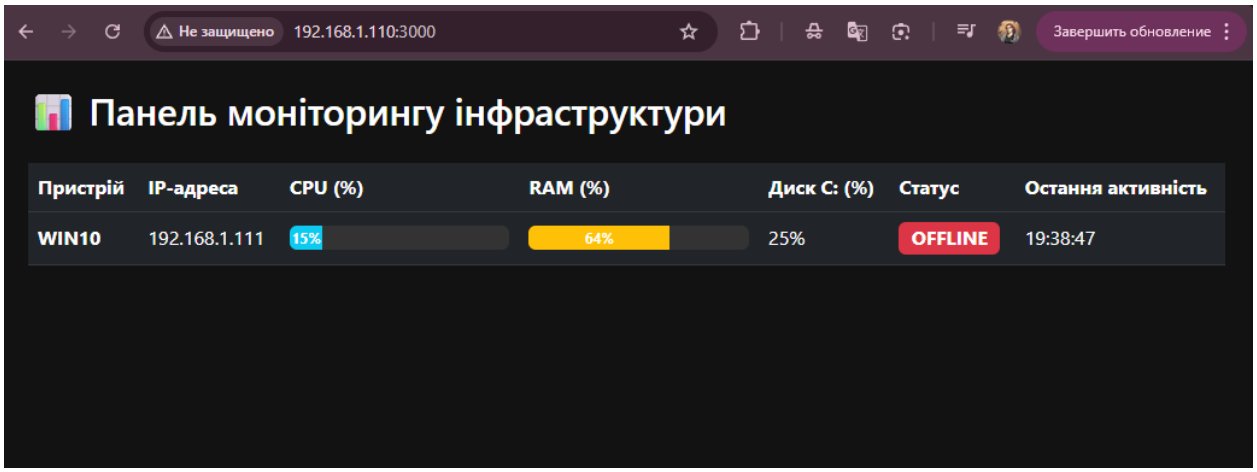

Рисунок 3.11 — Запуск API-сервера на порту 3000

Джерело: розроблено автором

Отже, серверне API забезпечує основну логіку роботи системи: приймання даних, перевірку токена, запис у базу даних і видачу даних для вебпанелі (рис. 3.11). Його реалізація є центральним етапом практичної частини дипломної роботи.

3.5 Реалізація клієнтського PowerShell-агента

Клієнтський агент є програмним компонентом, який виконується на Windows-пристроях і забезпечує збір показників стану системи. У межах дипломної роботи агент реалізовано у вигляді PowerShell-скрипта monitor.ps1. Такий підхід є доцільним, оскільки PowerShell є штатним засобом автоматизації в середовищі Windows і дозволяє отримувати системну інформацію, формувати HTTP-запити та передавати дані на серверне API.

PowerShell використовується для автоматизації адміністративних задач, роботи з об’єктами операційної системи та виконання сценаріїв керування. Для отримання даних про комп’ютерну систему у Windows можуть використовуватися WMI/CIM-класи, які надають доступ до відомостей про процесор, операційну систему, диски, мережеві адаптери та інші компоненти [24]. У цій роботі такі

можливості застосовано для збору показників CPU, RAM, Disk та IP-адреси клієнтського пристрою.

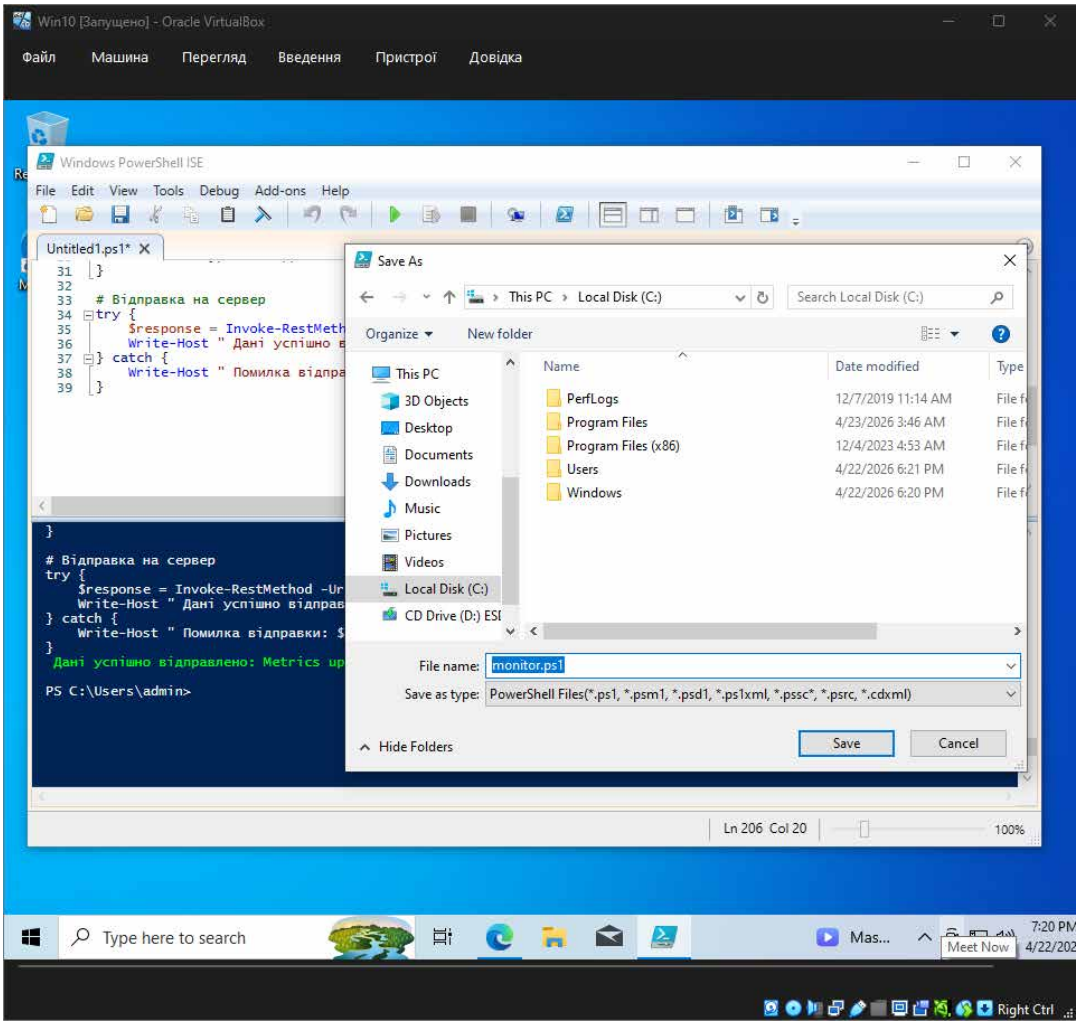

Рисунок 3.12 — Файл клієнтського агента monitor.ps1 на Windows-клієнті

Джерело: розроблено автором

На початку скрипта задаються адреса серверного API та токен доступу:

$apiUrl = http://192.168.1.110:3000/api/devices

$token = "ApiToken2026"

Адреса $apiUrl визначає, куди саме клієнтський агент буде надсилати зібрані показники. У разі перенесення стенду в іншу мережу IP-адресу сервера необхідно змінити. В інструкції з розгортання стенду зазначено, що після запуску Ubuntu Server потрібно визначити його нову IP-адресу командою ip a, а потім замінити стару адресу у файлі monitor.ps1 (рис. 3.12) на кожній Windows-машині .

Назва комп’ютера отримується через змінну середовища:

$name = $env:COMPUTERNAME

IP-адреса клієнтського пристрою визначається через WMI-запит до мережевого адаптера:

$ip = (Get-WmiObject Win32_NetworkAdapterConfiguration |
Where-Object { $_.IPEnabled -eq $true } |
Select-Object -First 1).IPAddress[0]

Завантаження процесора визначається за допомогою класу Win32_Processor, який у документації Microsoft описується як WMI-клас, що представляє процесорний пристрій у комп'ютері з операційною системою Windows [25]. У скрипті використовується властивість LoadPercentage, після чого значення усереднюється та округлюється.

$cpu = [math]::Round((Get-WmiObject win32_processor |
Measure-Object -Property LoadPercentage -Average).Average)

Використання оперативної пам'яті розраховується на основі загального обсягу доступної фізичної пам'яті та обсягу вільної пам'яті:

$os = Get-WmiObject win32_operatingsystem

*$ram = [math]::Round((($os.TotalVisibleMemorySize - $os.FreePhysicalMemory) /$os.TotalVisibleMemorySize) * 100)*

Заповнення диска визначається для системного диска C::

$diskInfo = Get-WmiObject Win32_LogicalDisk -Filter "DeviceID='C:'"

$disk = [math]::Round((($diskInfo.Size - $diskInfo.FreeSpace) / $diskInfo.Size) 100)*

Після отримання показників формується об'єкт із даними пристрою. Далі він перетворюється у формат JSON (рис. 3.13):

```
$body = @{
    name = $name
    ip = $ip
    cpu = $cpu
    ram = $ram
    disk = $disk
    status = "online"
} | ConvertTo-Json
```

Рисунок 3.13 – Вигляд JSON-файлу з даними пристрою

Джерело: розроблено автором

Для передавання токена доступу та вказання формату даних створюються HTTP-заголовки:

```
$headers = @{
   "Authorization" = "Bearer $token"
   "Content-Type" = "application/json"
}
```

Надсилання даних на сервер виконується командлетом Invoke-RestMethod. В офіційній документації Microsoft зазначено, що Invoke-RestMethod надсилає HTTP та HTTPS-запити до REST-вебслужб і повертає структуровані дані; для JSON-відповідей PowerShell може перетворювати отриманий вміст на об’єкти PowerShell [26]. У межах системи моніторингу цей командлет використовується для передавання POST-запиту на маршрут /api/devices.

```
try {
   $response = Invoke-RestMethod -Uri $apiUrl -Method Post -Body $body -Headers $headers
   Write-Host "Дані успішно відправлено: $($response.message)" -ForegroundColor Green
} catch {
   Write-Host "Помилка відправки: $_" -ForegroundColor Red
}
```

Лістинг 3.5 – Командлет для передавання POST-запиту на маршрут /api/devices

Джерело: розроблено автором

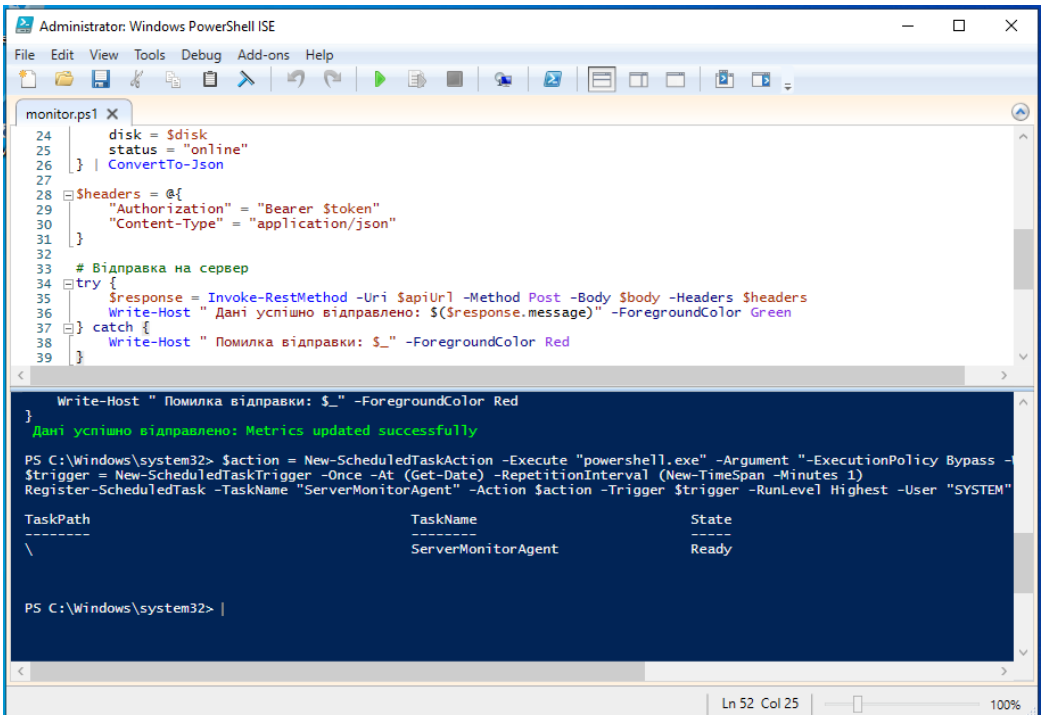

Рисунок 3.14 — Успішне передавання даних PowerShell-агентом на сервер

Джерело: розроблено автором

Для регулярного запуску агента використовується планувальник завдань Windows. Microsoft визначає Task Scheduler як засіб, що дає змогу автоматично виконувати рутинні задачі за встановленими умовами або тригерами [27]. У практичній реалізації завдання запускає файл monitor.ps1 щохвилини, завдяки чому дані у вебпанелі регулярно оновлюються без ручного запуску скрипта.

У наданій інструкції з розгортання стенду зазначено, що після редагування IP-адреси клієнти автоматично починають надсилати дані на сервер, оскільки завдання в Task Scheduler запускається щохвилини . Це забезпечує роботу системи в режимі, наближеному до реального часу.

Отже, PowerShell-агент виконує функцію джерела даних у системі моніторингу. Він збирає локальні показники Windows-пристрою, формує JSON-запит і передає його на серверне API з використанням токена доступу.

3.6 Реалізація вебпанелі моніторингу

Вебпанель є інтерфейсом користувача, через який адміністратор отримує інформацію про стан серверів і клієнтських пристроїв. Вона завершує ланцюг роботи системи: клієнтські агенти передають дані на сервер, сервер зберігає їх у базі даних, а вебпанель відображає ці дані у зрозумілому вигляді.

У межах практичної реалізації вебпанель розміщується на тому самому сервері Node.js, що й API. Для цього в серверному коді використовується роздача статичних файлів із папки public. У наданих матеріалах проєкту зазначено, що вебпанель повинна містити список пристроїв, статус, оновлення кожні 5 секунд і отримувати дані через JavaScript-запит fetch('/api/devices') .

Основна сторінка вебпанелі може бути реалізована у файлі index.html. Вона містить заголовок, таблицю пристроїв і місце для динамічного виведення даних, лістинг представлено у додатку В.

Для отримання даних із серверного API використовується Fetch API. У документації MDN зазначено, що Fetch API надає інтерфейс для отримання ресурсів, зокрема через мережу, і є гнучкішою заміною XMLHttpRequest [28]. У межах цієї системи метод fetch() використовується для звернення до маршруту /api/devices та отримання списку пристроїв у форматі JSON (додаток Г).

У цьому фрагменті функція loadDevices() виконує запит до API, отримує відповідь у форматі JSON, очищує таблицю та заповнює її актуальними даними з бази. Використання setInterval(loadDevices, 5000) забезпечує автоматичне оновлення даних кожні 5 секунд. Це відповідає вимогам практичної частини, де зазначено необхідність регулярного оновлення вебпанелі .

Для покращення сприйняття інформації вебпанель може використовувати стилізацію CSS (додаток Ґ). Наприклад, статус online можна виділяти зеленим кольором, а статус offline — червоним. Також доцільно візуально підсвічувати високі значення CPU, RAM або Disk, щоб адміністратор швидко помічав потенційно проблемні пристрої.

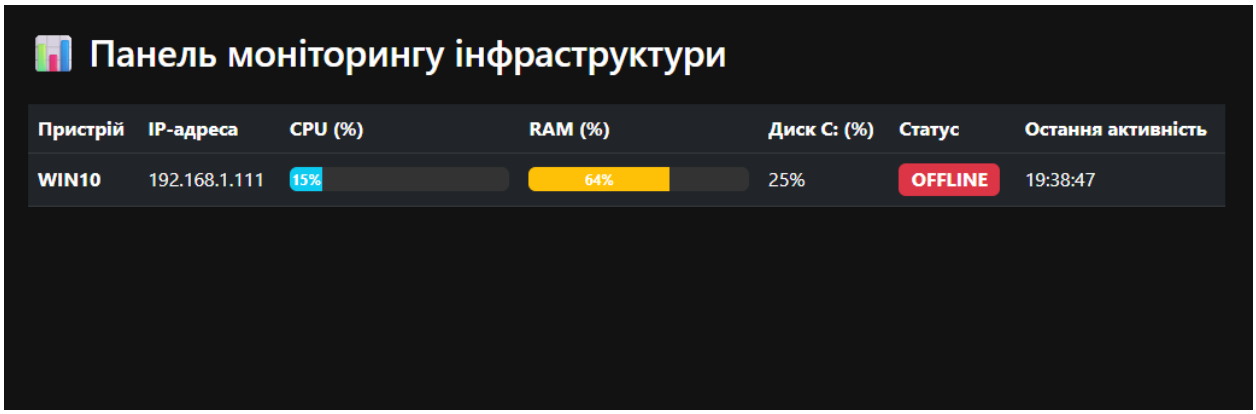

Рисунок 3.15 — Зовнішній вигляд вебпанелі моніторингу

Джерело: розроблено автором

У вебпанелі (рис. 3.15) відображаються всі основні показники, передбачені вимогами проєкту: назва пристрою, IP-адреса, CPU, RAM, Disk, статус і час останньої активності. Завдяки цьому адміністратор може в одному браузерному вікні переглядати поточний стан усіх підключених клієнтських машин.

3.7 Запуск системи та перевірка передавання даних

Після реалізації серверного API, бази даних, PowerShell-агента та вебпанелі виконується запуск усіх компонентів системи. Спочатку запускається Ubuntu Server, на якому повинні працювати MySQL і Node.js-застосунок. Далі запускаються Windows 10 і Windows Server, на яких виконується клієнтський агент monitor.ps1.

Серверний застосунок запускається з каталогу проєкту командою:

node server.js

Після запуску сервер повинен вивести повідомлення про роботу на порту 3000 (рис. 3.16):

API Server is running on port 3000

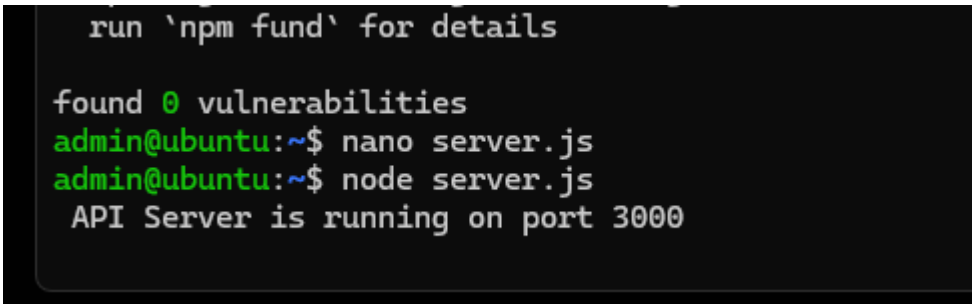

Рисунок 3.16 — Запуск серверного застосунку Node.js

Джерело: розроблено автором

Далі на клієнтських пристроях перевіряється виконання PowerShell-агента (рис. 3.17). Якщо IP-адреса сервера та токен задані правильно, агент надсилає POST-запит на сервер, а сервер записує або оновлює інформацію в таблиці devices.

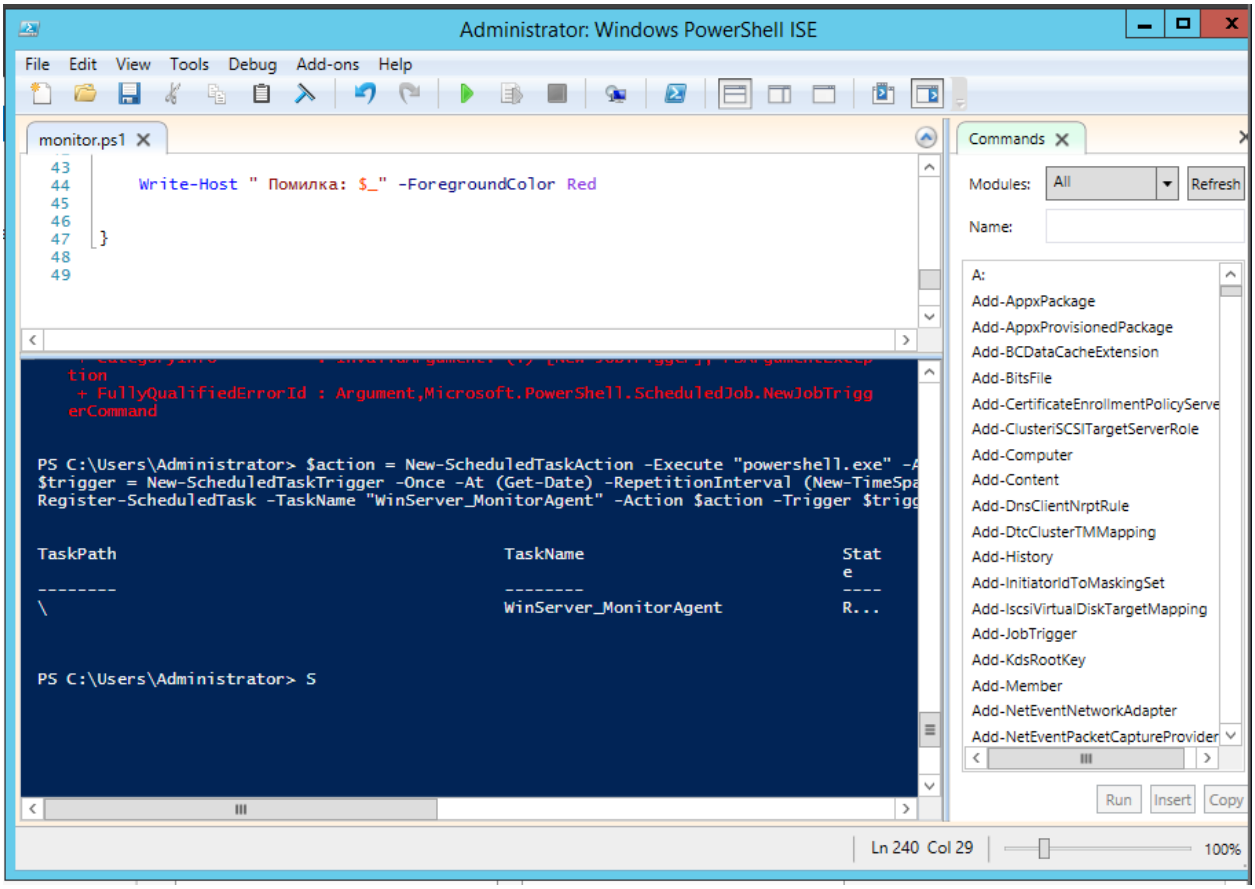

Рисунок 3.17 — Виконання monitor.ps1 на Windows Server

Джерело: розроблено автором

Після передавання даних можна перевірити вміст таблиці devices (рис. 3.18) у MySQL:

*SELECT * FROM devices;*

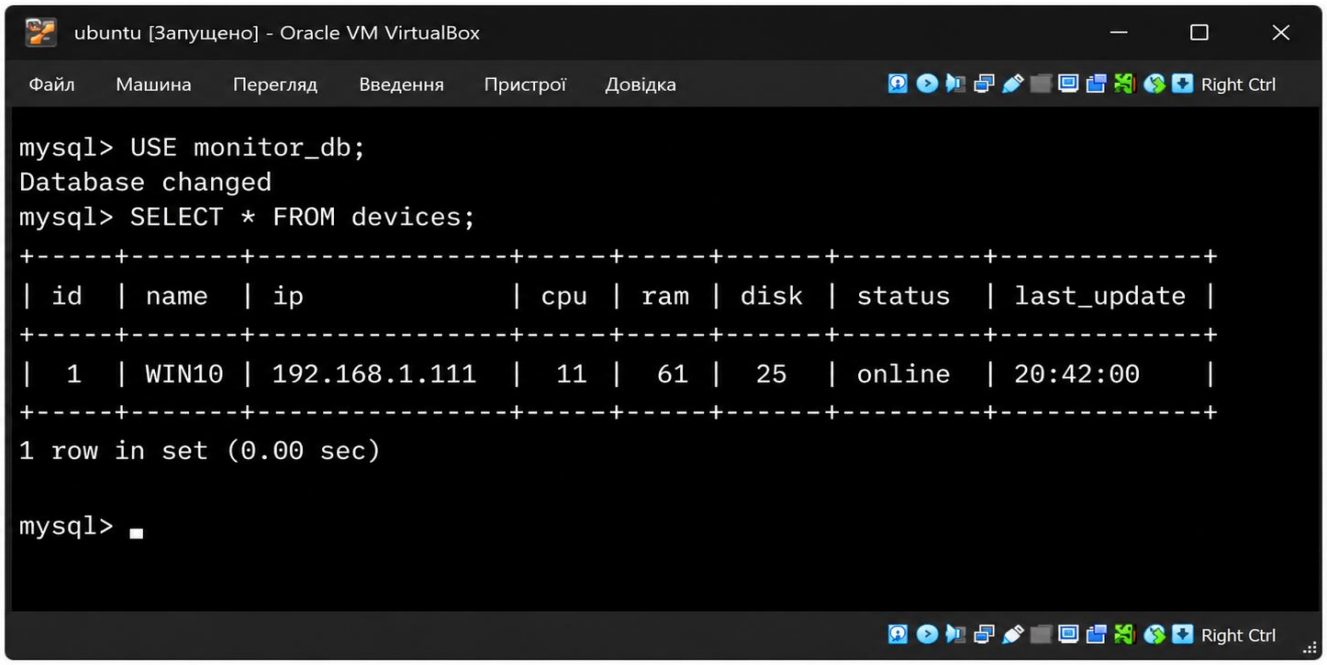

Рисунок 3.18 — Перегляд записів у таблиці devices після передавання даних

Джерело: розроблено автором

Якщо дані успішно записані в базу, наступним кроком є перевірка вебпанелі. Для цього на фізичному комп’ютері або на будь-якій машині в тій самій мережі потрібно відкрити браузер і перейти за адресою:

http://<IP-адреса Ubuntu Server>:3000

У наданій інструкції з розгортання стенду (додаток Б) зазначено, що після запуску всіх віртуальних машин вебпанель відкривається у браузері за адресою http://<НОВА_IP_АДРЕСА_UBUNTU>:3000, після чого користувач бачить панель моніторингу з актуальними даними (рис. 3.19) CPU, RAM і Disk від обох Windows-машин .

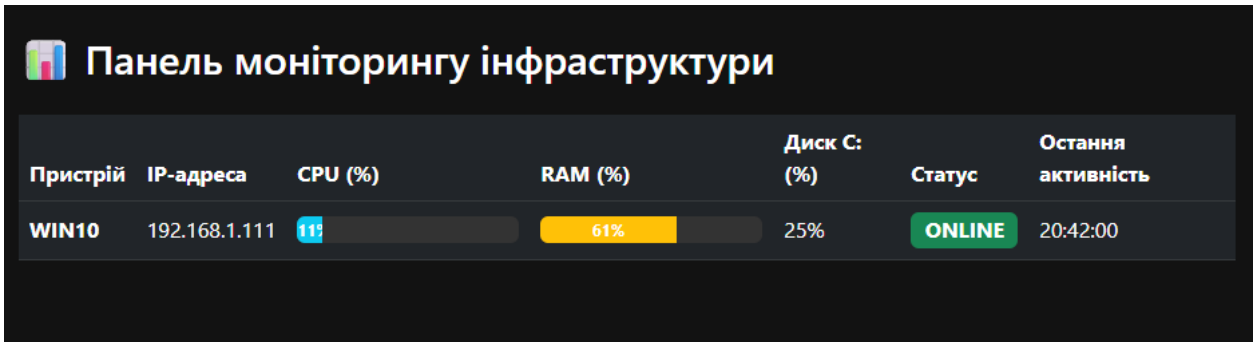

Рисунок 3.19 — Відображення даних Windows-клієнтів у вебпанелі моніторингу

Джерело: розроблено автором

На цьому етапі перевіряється повний цикл роботи системи (рис. 3.20):

1. Windows-клієнт збирає локальні показники.
2. PowerShell-агент формує JSON-запит.
3. Дані передаються на серверне API.
4. API перевіряє токен доступу.
5. Дані записуються або оновлюються в MySQL.
6. Вебпанель отримує інформацію через GET /api/devices.
7. Адміністратор бачить актуальний стан пристроїв у браузері.

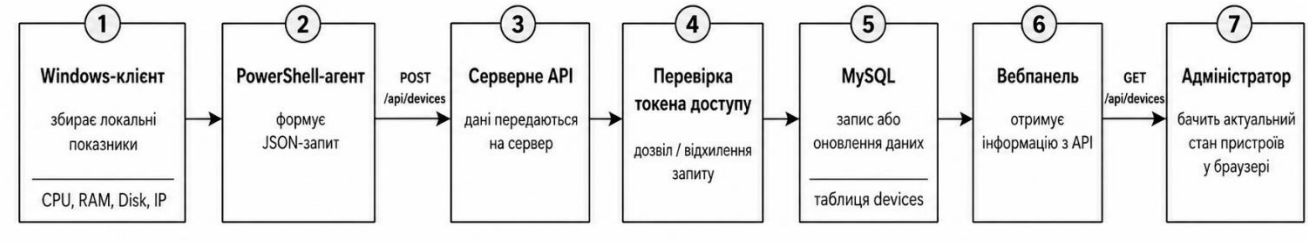

Рисунок 3.20 — Повний цикл передавання даних у системі моніторингу

Джерело: розроблено автором

У результаті запуску системи підтверджено, що клієнтські пристрої можуть передавати свої показники на Ubuntu Server, серверна частина записує дані до бази MySQL, а вебпанель відображає актуальний стан пристроїв у браузері.

3.8 Висновки до розділу 3

У третьому розділі було описано практичну реалізацію вебпанелі моніторингу серверів і пристроїв. Для тестування системи створено віртуальний лабораторний стенд на базі Oracle VM VirtualBox, який включає Ubuntu Server, Windows 10 і Windows Server. Для забезпечення мережевої взаємодії між машинами використано режим Bridged Adapter.

На Ubuntu Server було встановлено необхідне серверне програмне забезпечення: Node.js, npm і MySQL Server. Створено базу даних monitor_db і таблицю devices, у якій зберігаються назва пристрою, IP-адреса, показники CPU, RAM, Disk, статус і час останнього оновлення.

Серверне API реалізовано на основі Node.js та Express. Воно приймає дані від клієнтських агентів, перевіряє токен доступу, записує або оновлює інформацію в MySQL і надає вебпанелі список пристроїв. Клієнтський агент реалізовано у вигляді PowerShell-скрипта monitor.ps1, який збирає локальні показники Windows-пристрою та передає їх на сервер через HTTP POST-запит.

Вебпанель реалізовано з використанням HTML, CSS і JavaScript. Вона отримує дані з API за допомогою Fetch API, відображає список пристроїв і автоматично оновлює інформацію кожні 5 секунд. У результаті реалізовано повний цикл роботи системи моніторингу: від збору даних на клієнті до їх відображення у браузері адміністратора.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ

4.1 Мета та завдання тестування системи

Після реалізації основних компонентів системи моніторингу необхідно виконати її тестування. Тестування дає змогу перевірити, чи коректно взаємодіють між собою клієнтські агенти, серверне API, база даних і вебпанель. Основна мета тестування полягає у підтвердженні працездатності системи в умовах віртуального лабораторного стенду та виявленні можливих помилок у передаванні, збереженні й відображенні даних.

У межах цієї дипломної роботи тестування проводилося для системи, що складається з трьох віртуальних машин: Ubuntu Server, Windows 10 і Windows Server. Ubuntu Server виконує роль центрального сервера, на якому розміщені API, база даних MySQL і вебпанель. Windows 10 і Windows Server виконують роль клієнтів, які передають показники CPU, RAM і Disk на сервер через PowerShell-агент.

Тестування системи є важливим не лише з погляду перевірки програмної реалізації, а й з погляду забезпечення стабільності інформаційної інфраструктури. Закон України «Про основні засади забезпечення кібербезпеки України» визначає організаційні та правові основи захисту інтересів людини, суспільства й держави у кіберпросторі, а також напрями державної політики у сфері кібербезпеки [29]. У контексті цієї роботи система моніторингу не є повноцінним засобом кіберзахисту, однак вона виконує допоміжну функцію контролю стану вузлів інформаційної інфраструктури.

Основними завданнями тестування є:

- перевірити мережеву доступність між Ubuntu Server, Windows 10 і Windows Server;
- перевірити коректність запуску серверного API;
- перевірити роботу маршруту POST /api/devices;
- перевірити роботу маршруту GET /api/devices;

- перевірити запис даних у таблицю devices;
- перевірити виконання PowerShell-агента на Windows-клієнтах;
- перевірити автоматичне оновлення даних у вебпанелі;
- перевірити базовий захист API через токен доступу;
- оцінити переваги та обмеження розробленої системи.

Таблиця 4.1 — План тестування системи моніторингу

№	Етап тестування	Очікуваний результат
1	Перевірка IP-адрес віртуальних машин	Усі машини мають IP-адреси в одній підмережі
2	Перевірка доступності Ubuntu Server	Windows-клієнти можуть звертатися до сервера
3	Запуск Node.js API	Сервер працює на порту 3000
4	Надсилання POST-запиту	Дані приймаються API та записуються в MySQL
5	Отримання GET-запиту	API повертає список пристроїв у JSON
6	Виконання monitor.ps1	Клієнт передає CPU, RAM, Disk, IP і статус
7	Перевірка вебпанелі	Дані відображаються у браузері
8	Перевірка токена	API відхиляє запити з неправильним токеном

Джерело: розроблено автором

4.2 Перевірка мережевої взаємодії віртуальних машин

Першим етапом тестування є перевірка мережевої взаємодії між віртуальними машинами. Оскільки в системі використовується клієнт-серверна архітектура, Windows-клієнти повинні мати можливість звертатися до Ubuntu Server за IP-адресою. Якщо мережевий зв’язок між машинами відсутній, клієнтські агенти не зможуть передавати метрики на серверне API.

У практичній реалізації всі віртуальні машини підключені через режим **Bridged Adapter**. Як зазначено в документації Oracle VirtualBox, bridged networking дозволяє віртуальній машині підключатися до мережі через фізичний мережевий інтерфейс хостової системи [17]. У наданій інструкції з розгортання стенду також

зазначено, що для кожної машини потрібно вибрати режим **Мережевий міст** і прив’язати його до актуального фізичного адаптера комп’ютера .

Для перевірки IP-адреси Ubuntu Server використовується команда:

ip a

Для перевірки IP-адрес Windows-клієнтів можна використати команду:

ipconfig

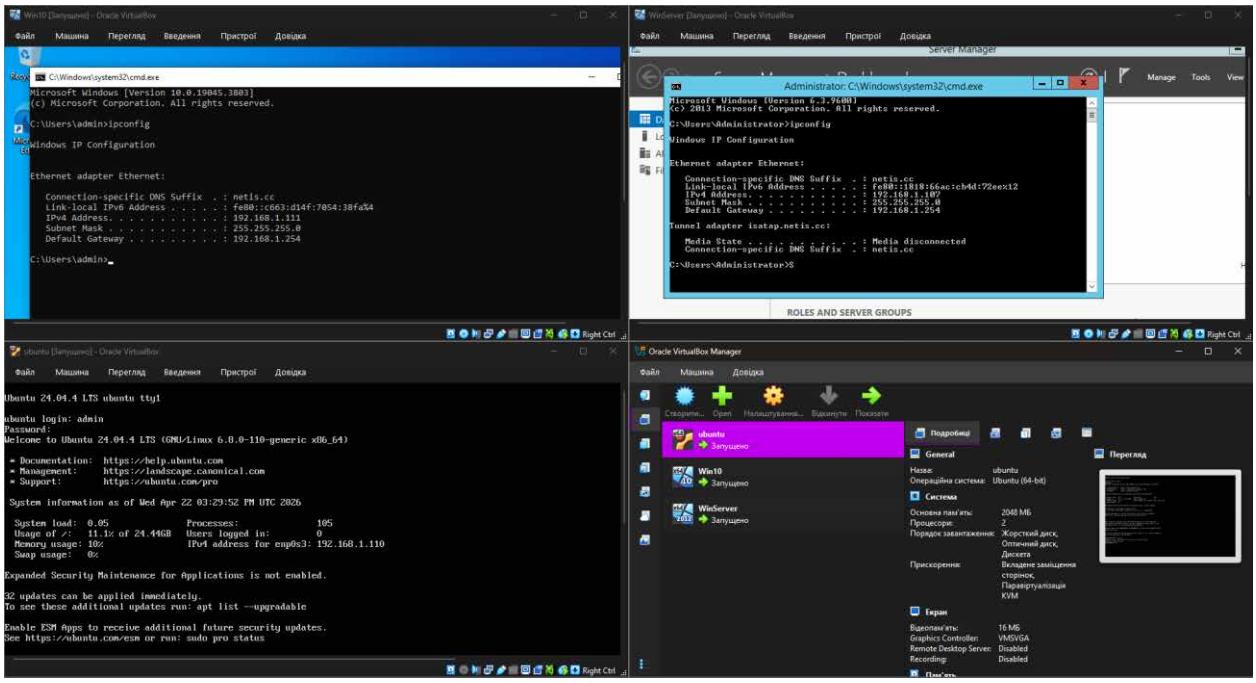

Рисунок 4.1 — Перевірка IP-адрес усіх віртуальних машин

Джерело: розроблено автором

Після визначення IP-адрес (рис. 4.1) виконується перевірка доступності сервера з клієнтських машин. Для цього можна використати команду:

ping <IP-адреса Ubuntu Server>

Перед перевіркою доступності Ubuntu сервера виконується підготовка сервера, а саме встановлюється статична IP адреса (рис. 4.2) для простішої доступності до даного вузла.

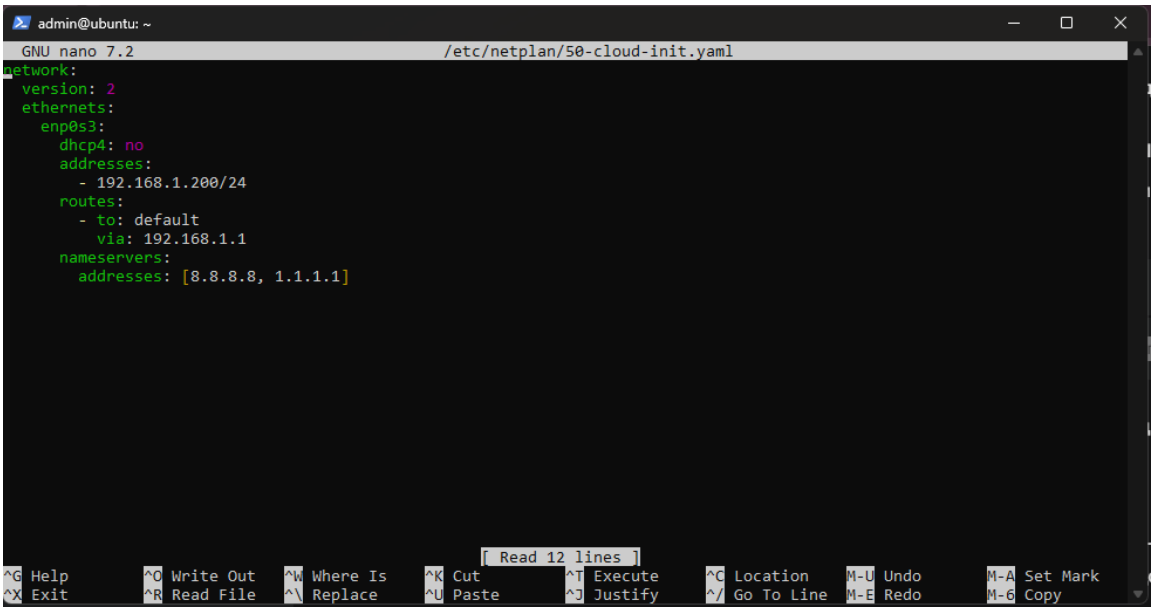

Рисунок 4.2 – Встановлення статичної IP-адреси сервера

Джерело: розроблено автором

Якщо мережеве налаштування виконано правильно, Windows-клієнт отримає відповіді від Ubuntu Server. Це підтверджує, що машини знаходяться в одній мережі та можуть взаємодіяти між собою. У матеріалах практичної реалізації зазначено, що всі три машини знаходяться в одній підмережі 192.168.1.X і використовують один шлюз, що підтверджує коректну роботу мережевого мосту .

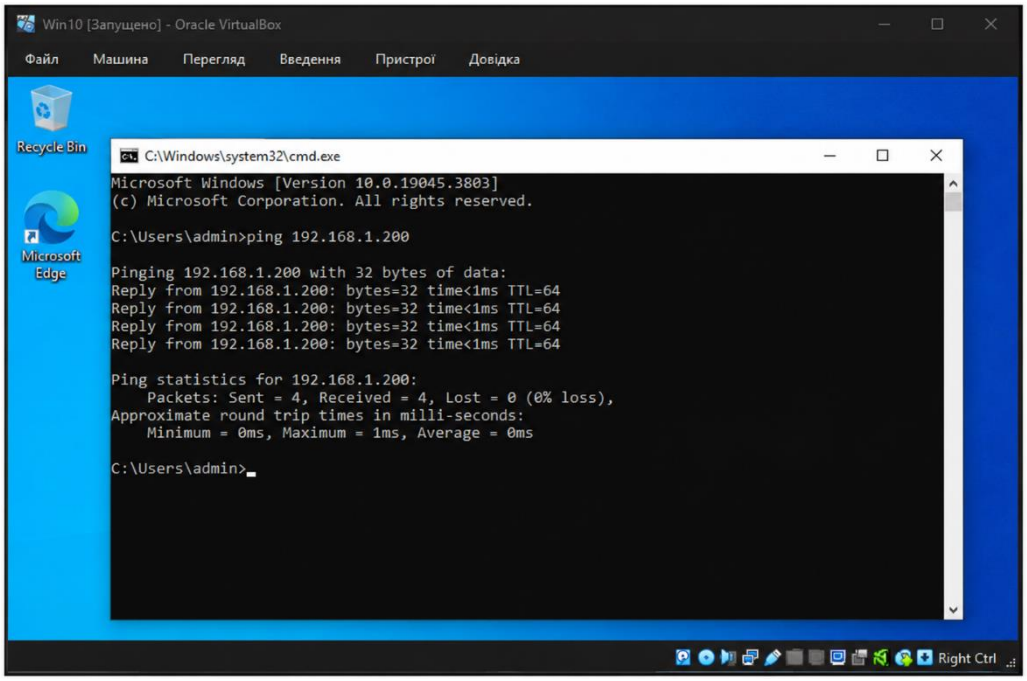

Рисунок 4.3 — Перевірка доступності Ubuntu Server командою ping

Джерело: розроблено автором

Результатом цього етапу є підтвердження мережевої доступності між усіма компонентами стенду. Це є обов'язковою умовою для подальшого тестування API, бази даних і вебпанелі.

4.3 Перевірка роботи серверного API

Наступним етапом є перевірка роботи серверного API. API виконує роль центрального компонента системи, тому його коректна робота визначає можливість приймання, збереження та відображення даних.

Після запуску серверного застосунку командою:

node server.js

у консолі має з'явитися повідомлення про запуск сервера на порту 3000. Після цього можна перевірити доступність API через браузер або спеціальний інструмент для HTTP-запитів.

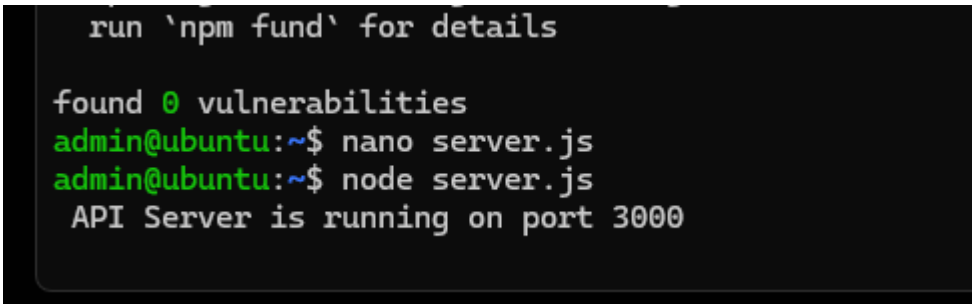

Рисунок 4.4 — Запуск серверного API на Ubuntu Server

Джерело: розроблено автором

Для перевірки маршруту *GET /api/devices* у браузері або Postman виконується запит:

http://<IP-адреса Ubuntu Server>:3000/api/devices

Якщо API працює коректно, сервер повертає JSON-масив із записами пристроїв. Якщо в базі ще немає пристроїв, API може повернути порожній масив. Формат відповіді JSON є доцільним для такої системи, оскільки JSON широко використовується для обміну структурованими даними між клієнтами та вебсерверами [14].

Для перевірки маршруту POST /api/devices (рис. 4.5) можна використати Postman або інший HTTP-клієнт. HTTP-коди стану дають змогу визначити результат оброблення запиту: успішне виконання, помилку клієнта або помилку сервера. У документації MDN зазначено, що HTTP response status codes показують, чи був конкретний HTTP-запит успішно завершений, і поділяються на кілька класів відповідей [30].

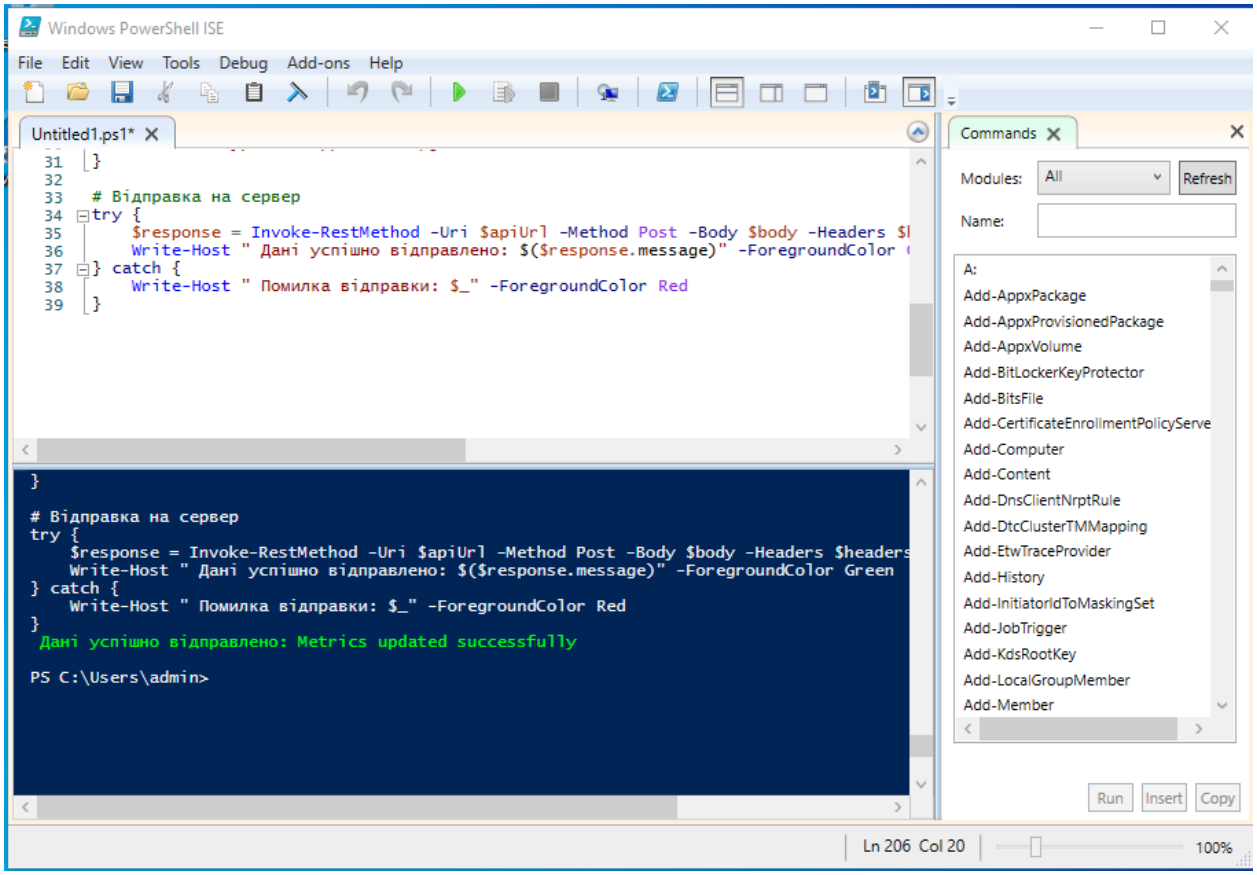

Рисунок 4.5 – Приклад заголовків запиту

Джерело: розроблено автором

Приклад заголовків запиту (рис. 4.6):

```
Authorization: Bearer ApiToken2026
Content-Type: application/json
```

Рисунок 4.6 – Приклад заголовків запиту

Джерело: розроблено автором

Очікуваним результатом є відповідь сервера про успішне оновлення метрик. Якщо токен неправильний або відсутній, сервер повинен повернути помилку доступу. Така поведінка підтверджує, що механізм базової перевірки токена працює коректно.

4.4 Перевірка запису даних у базу MySQL

Після успішного POST-запиту необхідно перевірити, чи були дані записані в базу MySQL (рис. 4.7). Для цього виконується вхід у консоль MySQL:

sudo mysql

Далі потрібно перейти до бази даних:

USE monitor_db;

Після цього виконується запит:

*SELECT * FROM devices;*

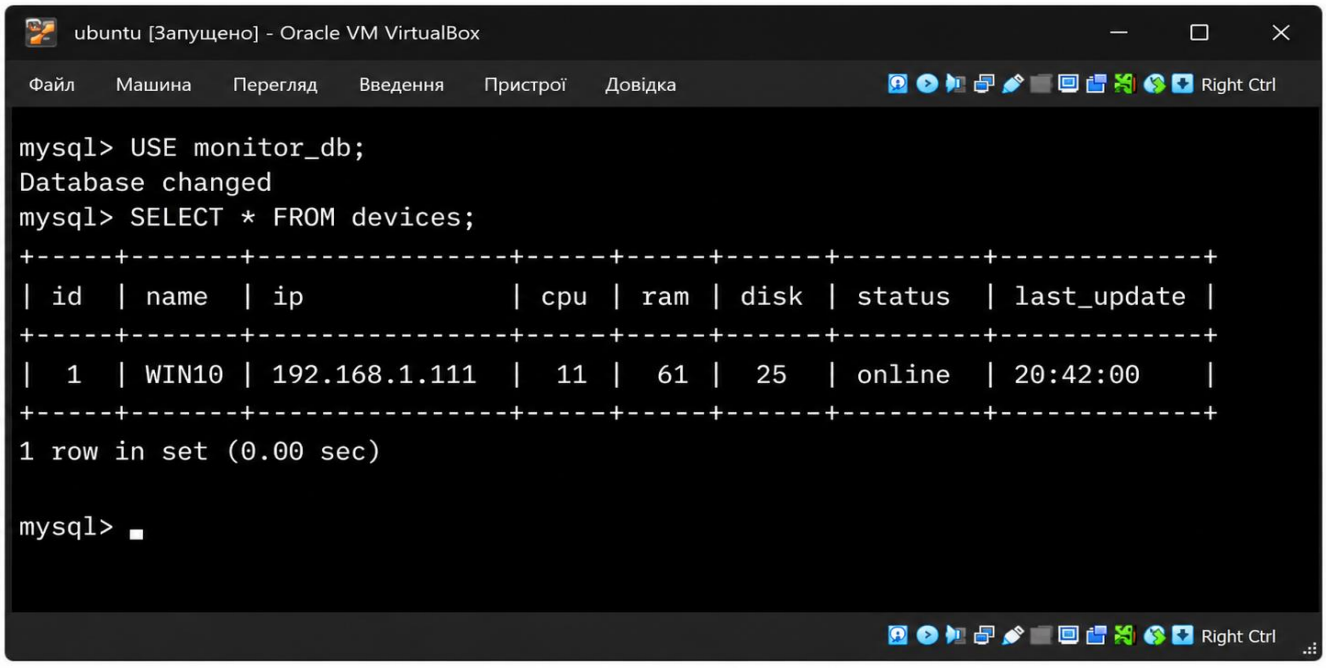

Рисунок 4.7 — Перегляд записів у таблиці devices

Джерело: розроблено автором

У таблиці повинні з’явитися записи, які відповідають даним, переданим через API. Для тестового запису це може бути пристрій TEST-CLIENT, а для реальних

клієнтів — назви Windows 10 і Windows Server. Наявність таких записів підтверджує, що серверне API коректно взаємодіє з базою даних.

Окремо перевіряється механізм оновлення записів. Якщо один і той самий клієнт повторно передає дані, система не повинна створювати зайві дублікати. Для цього в API використовується конструкція INSERT ... ON DUPLICATE KEY UPDATE, яка дозволяє оновити запис при конфлікті унікального ключа [21]. Такий підхід є важливим для системи моніторингу, оскільки клієнтські агенти передають дані регулярно.

Результатом цього етапу є підтвердження, що база даних правильно зберігає й оновлює інформацію про пристрої.

4.5 Перевірка роботи клієнтського PowerShell-агента

Після перевірки API та бази даних виконується тестування клієнтського PowerShell-агента. Агент запускається на Windows 10 і Windows Server та збирає локальні показники пристрою. Його робота полягає у визначенні назви комп’ютера, IP-адреси, завантаження CPU, використання RAM, заповнення диска та передаванні цих даних на сервер.

Для ручної перевірки скрипт можна запустити з PowerShell (рис. 4.8):

.\monitor.ps1

Якщо скрипт виконано успішно, у консолі має з’явитися повідомлення про відправлення даних. У попередньому розділі було зазначено, що для надсилання даних використовується командлет Invoke-RestMethod, який надсилає HTTP/HTTPS-запити до REST-вебслужб [26].

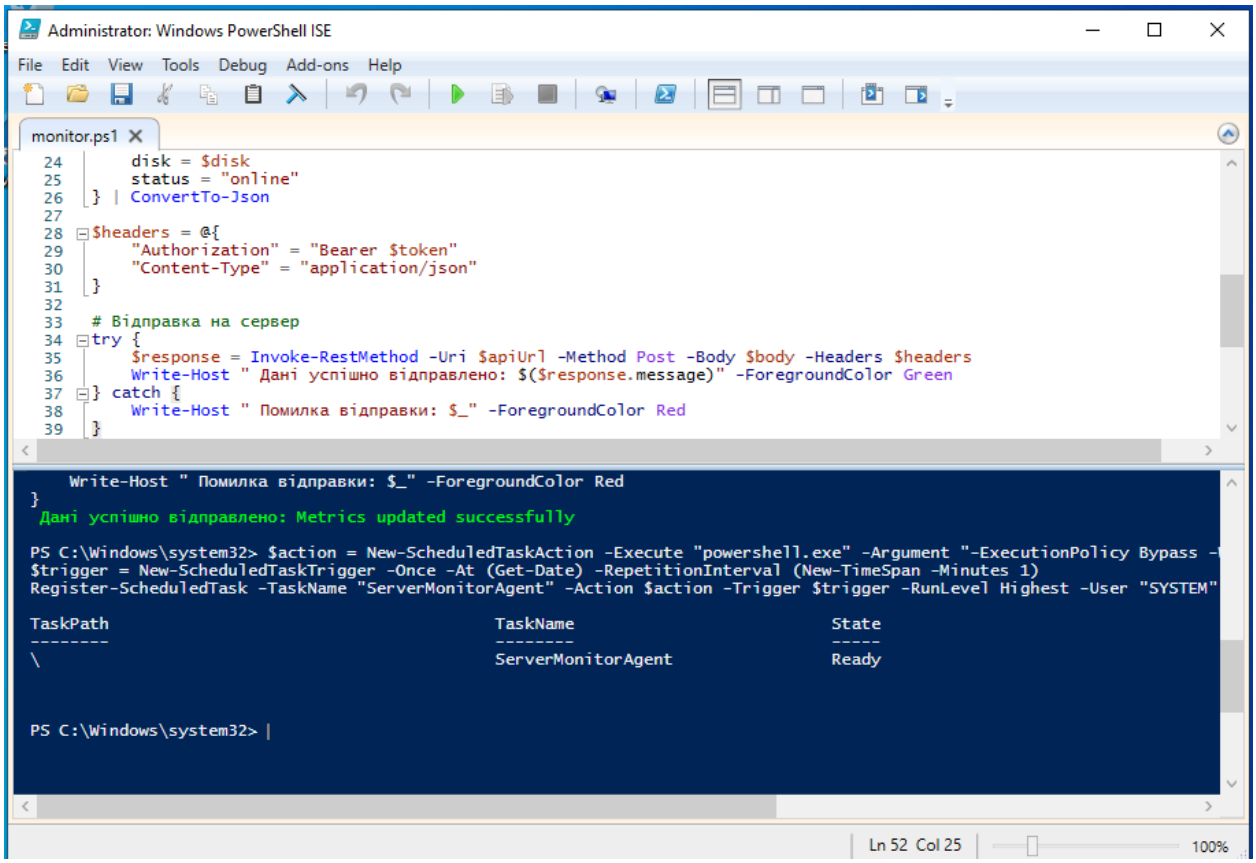

Рисунок 4.8 — Ручний запуск monitor.ps1 на Windows 10

Джерело: розроблено автором

Після запуску скрипта потрібно перевірити таблицю devices у MySQL і вебпанель. Якщо все працює коректно, у таблиці та в інтерфейсі мають з’явитися або оновитися записи відповідних Windows-клієнтів.

Для автоматичної роботи агента використовується Task Scheduler. Microsoft описує Task Scheduler як компонент Windows, який дає змогу автоматично виконувати задачі у визначений час або за певними умовами [27]. У практичній реалізації завдання запускає monitor.ps1 щохвилини, що забезпечує регулярне оновлення інформації у вебпанелі.

У результаті цього тесту підтверджено, що клієнтський агент може працювати як у ручному режимі, так і в автоматичному режимі за розкладом.

4.6 Перевірка роботи вебпанелі моніторингу

Останнім функціональним етапом тестування є перевірка вебпанелі (рис. 4.9). Для цього у браузері відкривається адреса:

http://<IP-адреса Ubuntu Server>:3000

У наданій інструкції з розгортання стенду (додаток Б) зазначено, що після запуску всіх віртуальних машин користувач відкриває браузер на фізичному комп'ютері та переходить за адресою http://<НОВА_IP_АДРЕСА_UBUNTU>:3000, після чого бачить вебпанель з актуальними даними CPU, RAM і Disk від обох Windows-машин.

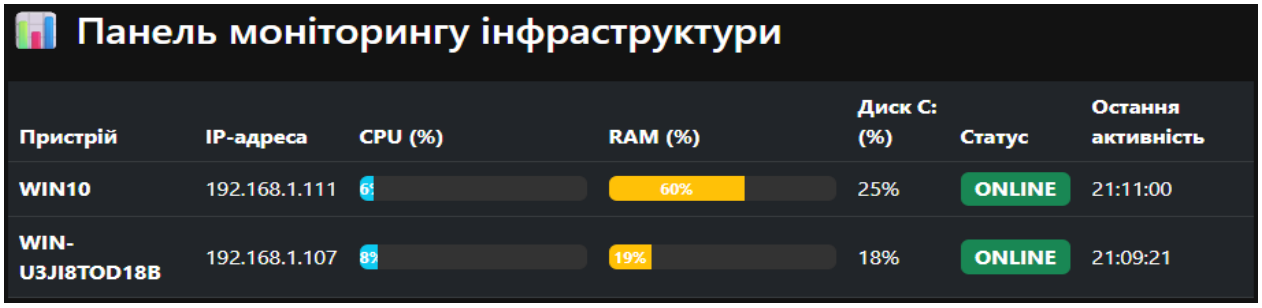

Рисунок 4.9 — Вебпанель моніторингу у браузері

Джерело: розроблено автором

На вебпанелі повинні відображатися такі дані:

- назва пристрою;
- IP-адреса;
- завантаження CPU;
- використання RAM;
- заповнення Disk;
- статус пристрою;
- час останнього оновлення.

Вебпанель отримує дані з маршруту GET /api/devices за допомогою Fetch API. Як зазначено в документації MDN, Fetch API використовується для отримання ресурсів через мережу [28]. У системі моніторингу він застосовується для

регулярного отримання JSON-даних із сервера та оновлення таблиці пристроїв у браузері.

Для перевірки автоматичного оновлення можна змінити навантаження на клієнтському пристрої або повторно запустити PowerShell-агент. Через декілька секунд дані у вебпанелі повинні змінитися. Якщо використано інтервал оновлення 5 секунд, зміни мають відображатися без ручного перезавантаження сторінки.

Результатом цього етапу є підтвердження того, що вебпанель коректно отримує дані з API, відображає їх у таблиці та оновлює інформацію в режимі, наближеному до реального часу.

4.7 Перевірка базового захисту API

Окремим етапом тестування є перевірка базового захисту API. У системі використовується токен доступу ApiToken2026, який клієнтський агент передає в HTTP-заголовку Authorization. Якщо токен відсутній або неправильний, сервер повинен відхилити запит.

Для перевірки можна виконати POST-запит без заголовка авторизації або з неправильним токеном. Очікуваним результатом є відповідь із кодом помилки доступу, наприклад 403 Forbidden.

Питання автентифікації та авторизації є важливим для API. OWASP API Security Top 10 окремо виділяє ризик порушеної автентифікації, коли API не перевіряє належним чином облікові дані або токени доступу [31]. У межах розробленого стенду застосовано просту перевірку токена, що не є достатнім рішенням для промислового використання, але демонструє базовий принцип контролю доступу до API.

В українському нормативному полі питання реагування на кіберподії також розглядається як важливий елемент забезпечення кібербезпеки. Держспецзв’язку зазначає, що методичні рекомендації щодо реагування на події в кіберпросторі визначають перелік заходів кіберзахисту, яких можуть послідовно вживати суб’єкти забезпечення кібербезпеки [32]. У контексті цієї дипломної роботи базова

перевірка токена й моніторинг стану пристроїв можуть розглядатися як елементи мінімального контролю доступу та спостереження за станом інфраструктури.

Результатом цього етапу є підтвердження, що API не приймає дані від клієнтів без правильного токена, а отже має базовий механізм обмеження доступу.

4.8 Аналіз отриманих результатів

За результатами тестування встановлено, що розроблена система виконує основні функції, визначені в постановці задачі. Клієнтські пристрої на базі Windows 10 і Windows Server успішно збирають показники стану системи та передають їх на сервер. Серверне API приймає дані, перевіряє токен доступу, записує або оновлює інформацію в базі даних MySQL. Вебпанель отримує дані з API та відображає їх у браузері адміністратора.

Основними результатами практичної реалізації є:

- створено віртуальний лабораторний стенд із трьох машин;
- налаштовано мережеву взаємодію через Bridged Adapter;
- розгорнуто Ubuntu Server як центральний сервер системи;
- встановлено Node.js, npm і MySQL Server;
- створено базу даних monitor_db і таблицю devices;
- реалізовано серверне API на Node.js та Express;
- реалізовано перевірку токена доступу;
- реалізовано PowerShell-агент для Windows-клієнтів;
- реалізовано вебпанель з автоматичним оновленням даних;
- перевірено повний цикл передавання даних від клієнта до вебінтерфейсу.

Таблиця 4.2 — Результати тестування системи

Етап	Результат	Статус
Перевірка мережі	Машини знаходяться в одній підмережі	Виконано
Запуск API	Сервер працює на порту 3000	Виконано

POST-запит	Дані приймаються та записуються в MySQL	Виконано
GET-запит	API повертає список пристроїв	Виконано
PowerShell-агент	Збирає та передає CPU, RAM, Disk	Виконано
Task Scheduler	Агент запускається автоматично	Виконано
Вебпанель	Дані відображаються у браузері	Виконано
Перевірка токена	Запити без правильного токена відхиляються	Виконано

Джерело: розроблено автором

Розроблена система має такі переваги:

- проста архітектура, зрозуміла для навчального проєкту;
- використання поширених технологій;
- можливість запуску в повністю віртуальному середовищі;
- централізоване збереження показників у базі даних;
- наявність вебпанелі для перегляду стану пристроїв;
- можливість автоматичного оновлення даних;
- базовий механізм обмеження доступу до API.

Разом із тим система має певні обмеження. По-перше, вона зберігає переважно актуальний стан пристроїв, а не повну історію змін. Для побудови повноцінних графіків за тривалий період доцільно додати окрему таблицю історії метрик. По-друге, використаний механізм токена є спрощеним і не забезпечує повноцінного рівня безпеки для промислового використання. По-третє, клієнтський агент реалізовано лише для Windows-пристроїв, тому для Linux-клієнтів потрібно створити окремий агент.

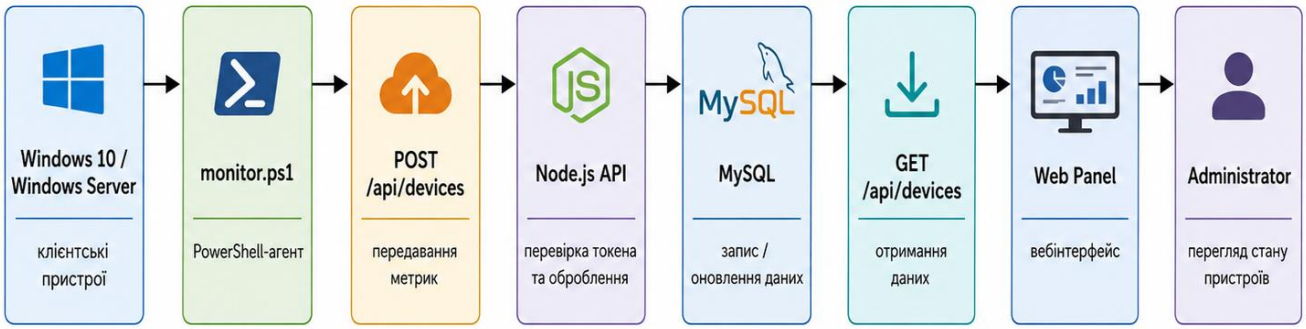

Рисунок 4.15 — Підсумкова схема роботи реалізованої системи моніторингу

Джерело: розроблено автором

4.9 Напрями подальшого вдосконалення системи

Попри те, що розроблена система виконує основні функції моніторингу, вона може бути вдосконалена в майбутньому. Насамперед доцільно реалізувати збереження історії метрик. Для цього можна створити окрему таблицю metrics_history, у якій кожен запис міститиме назву пристрою, часову мітку та значення CPU, RAM і Disk. Це дозволить будувати графіки зміни навантаження за день, тиждень або місяць.

Другим напрямом розвитку є додавання графіків у вебпанель. Для цього можна використати бібліотеку Chart.js, яка підтримує побудову різних типів графіків і може працювати з даними, отриманими через API [16]. Графіки зроблять вебпанель більш наочною та дозволять адміністратору оцінювати не лише поточний стан пристрою, а й динаміку змін.

Третім напрямом є реалізація системи сповіщень. Наприклад, якщо CPU, RAM або Disk перевищують визначені порогові значення, система може створювати подію або надсилати повідомлення адміністратору. Такий функціонал наблизить систему до професійних засобів моніторингу, у яких важливу роль відіграють тригери та повідомлення.

Четвертим напрямом є вдосконалення безпеки. Замість одного статичного токена можна реалізувати авторизацію користувачів, окремі токени для кожного клієнта, HTTPS-з’єднання та журналювання невдалих спроб доступу. Це особливо важливо, якщо система буде використовуватися не лише в навчальному стенді, а й у реальному середовищі.

П’ятим напрямом є створення клієнтського агента для Linux-пристроїв. Оскільки API приймає універсальні JSON-дані, до системи можна додати Linux-агент на Bash або Python, який передаватиме ті самі поля: name, ip, cpu, ram, disk і status.

Таким чином, розроблена система має потенціал для подальшого розвитку. Її базова архітектура дозволяє поступово додавати нові функції без повної зміни логіки роботи.

4.10 Висновки до розділу 4

У четвертому розділі було проведено тестування розробленої системи моніторингу серверів і пристроїв. Перевірено мережеву взаємодію між віртуальними машинами, запуск серверного API, роботу маршрутів POST /api/devices і GET /api/devices, запис даних у MySQL, виконання PowerShell-агента, автоматичний запуск через Task Scheduler і відображення даних у вебпанелі.

За результатами тестування підтверджено, що система виконує основні функції: збирає дані з Windows-клієнтів, передає їх на Ubuntu Server, зберігає у базі даних і відображає у браузері адміністратора. Також перевірено базовий механізм захисту API через токен доступу.

Розроблена система є працездатним навчальним стендом, який демонструє принципи централізованого моніторингу IT-інфраструктури. Вона може бути використана для навчальних і демонстраційних цілей, а також може бути розширена шляхом додавання історії метрик, графіків, сповіщень, авторизації користувачів і підтримки Linux-клієнтів.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто, спроєктовано та реалізовано систему моніторингу серверів і клієнтських пристроїв із використанням вебпанелі, серверного API, бази даних і клієнтських агентів. Основною метою роботи було створення програмного стенду, який забезпечує автоматизований збір, передавання, збереження та відображення основних показників стану комп'ютерних систем у локальній мережі.

У першому розділі було проаналізовано предметну область моніторингу IT-інфраструктури. Визначено, що контроль стану серверів і клієнтських пристроїв є важливим елементом адміністрування інформаційних систем, оскільки дозволяє своєчасно виявляти перевантаження ресурсів, втрату доступності пристроїв і потенційні проблеми в роботі локальної мережі. Було розглянуто основні показники, які доцільно контролювати в межах системи моніторингу: завантаження центрального процесора, використання оперативної пам'яті, заповнення дискового простору, IP-адресу, статус пристрою та час останнього оновлення. Також було проаналізовано існуючі засоби моніторингу, зокрема Zabbix, Prometheus, Grafana, Nagios і PRTG, та обґрунтовано доцільність створення власної спрощеної вебпанелі для навчального й демонстраційного стенду [3], [4], [5], [6], [7], [8].

У другому розділі було виконано проєктування системи моніторингу серверів і пристроїв. Визначено, що для реалізації поставленої задачі найдоцільнішою є клієнт-серверна архітектура, у якій Windows-пристрої виконують роль клієнтів, Ubuntu Server — роль центрального сервера, MySQL — роль сховища даних, а вебпанель — роль інтерфейсу адміністратора. Було обґрунтовано вибір технологій: Node.js і Express для реалізації серверного API, MySQL для збереження даних, PowerShell для клієнтського агента, HTML, CSS і JavaScript для вебпанелі. Такий набір технологій дозволив реалізувати всі основні функції системи без використання надмірно складних корпоративних платформ [9], [10], [11], [12], [13], [14].

У межах проєктування було визначено структуру бази даних monitor_db і таблиці devices, яка містить поля для збереження назви пристрою, IP-адреси, показників CPU, RAM, Disk, статусу та часу останнього оновлення. Також було спроєктовано серверне API з маршрутами POST /api/devices для приймання даних від клієнтів і GET /api/devices для передавання даних вебпанелі. Окремо було визначено логіку роботи клієнтського агента, який збирає локальні показники Windows-пристрою та передає їх на сервер у форматі JSON.

У третьому розділі було виконано практичну реалізацію системи. Для тестування створено віртуальний лабораторний стенд на базі Oracle VM VirtualBox, який складається з трьох віртуальних машин: Ubuntu Server, Windows 10 і Windows Server. Для забезпечення мережевої взаємодії використано режим Bridged Adapter, що дозволило всім машинам перебувати в одній локальній мережі [17]. На Ubuntu Server було встановлено Node.js, npm і MySQL Server, створено базу даних monitor_db, таблицю devices і окремого користувача для підключення API до бази даних [18], [19], [20].

Серверне API було реалізовано на основі Node.js та Express. Воно забезпечує приймання POST-запитів від клієнтських агентів, перевірку токена доступу, запис або оновлення даних у MySQL і видачу списку пристроїв для вебпанелі. Для зменшення кількості повторюваних записів у таблиці використано підхід оновлення наявного запису пристрою через конструкцію INSERT ... ON DUPLICATE KEY UPDATE [21]. Для базового обмеження доступу до API використано токен авторизації, що дозволило продемонструвати принцип контролю доступу до серверних маршрутів [23].

Клієнтський агент було реалізовано у вигляді PowerShell-скрипта monitor.ps1. Агент збирає назву комп’ютера, IP-адресу, показники CPU, RAM і Disk, формує JSON-об’єкт і передає його на сервер за допомогою HTTP POST-запиту. Для збору системних показників використано можливості WMI/CIM, а для передавання даних — командлет Invoke-RestMethod [24], [25], [26]. Для автоматичного запуску агента використано Task Scheduler, що забезпечує регулярне оновлення даних без ручного запуску скрипта [27].

Вебпанель було реалізовано з використанням HTML, CSS і JavaScript. Вона звертається до маршруту GET /api/devices, отримує дані у форматі JSON і відображає їх у таблиці. Для отримання даних із серверного API використано Fetch API [28]. Вебпанель відображає назву пристрою, IP-адресу, показники CPU, RAM і Disk, статус та час останнього оновлення. Автоматичне оновлення кожні 5 секунд дозволяє адміністратору бачити стан пристроїв у режимі, наближеному до реального часу.

У четвертому розділі було проведено тестування системи. Перевірено мережеву взаємодію між віртуальними машинами, запуск серверного API, роботу маршрутів POST /api/devices і GET /api/devices, запис даних у MySQL, виконання PowerShell-агента, автоматичний запуск через Task Scheduler, відображення даних у вебпанелі та базову перевірку токена доступу. За результатами тестування підтверджено, що система виконує поставлені функціональні вимоги: збирає дані з Windows-клієнтів, передає їх на Ubuntu Server, зберігає у базі даних і відображає у вебінтерфейсі адміністратора.

Практичне значення роботи полягає у створенні працездатного навчального стенду, який демонструє повний цикл роботи системи моніторингу IT-інфраструктури. Розроблена система може бути використана для навчальних, лабораторних і демонстраційних цілей, оскільки вона показує взаємодію між клієнтськими агентами, серверним API, базою даних і вебпанеллю. Крім того, система може бути основою для подальшого розвитку: додавання історії метрик, графіків, авторизації користувачів, системи сповіщень, підтримки Linux-клієнтів і розширених механізмів безпеки.

Отже, у результаті виконання кваліфікаційної роботи було досягнуто поставленої мети — розроблено вебпанель моніторингу серверів і клієнтських пристроїв, яка забезпечує автоматизований збір, збереження та відображення основних показників стану комп’ютерних систем у локальній мережі.

РЕКОМЕНДАЦІЇ

Розроблена система моніторингу може бути використана як навчальний стенд для демонстрації принципів побудови клієнт-серверних інформаційних систем. Вона є зручною для вивчення взаємодії між клієнтськими агентами, серверним API, базою даних і вебінтерфейсом. Завдяки використанню віртуального середовища система може бути розгорнута на одному фізичному комп’ютері без потреби у додатковому обладнанні.

Для практичного використання системи в реальному середовищі доцільно виконати її подальше вдосконалення. Насамперед рекомендується реалізувати збереження історії метрик. У поточній реалізації основна увага приділена відображенню актуального стану пристроїв, однак для аналізу навантаження за певний період доцільно створити окрему таблицю, наприклад metrics_history. У ній можна зберігати часові ряди показників CPU, RAM і Disk, що дозволить формувати графіки та аналізувати динаміку змін.

Другим напрямом удосконалення є додавання графіків у вебпанель. Для цього можна використати бібліотеку Chart.js, яка дозволяє будувати лінійні, стовпчикові та інші типи графіків [16]. Графічне подання даних підвищить наочність системи й дозволить адміністратору швидше оцінювати зміни навантаження.

Третім напрямом є реалізація системи сповіщень. Наприклад, якщо завантаження CPU або RAM перевищує встановлене порогове значення, система може відображати попередження у вебпанелі або надсилати повідомлення адміністратору. Такий функціонал наблизить систему до професійних засобів моніторингу, у яких важливу роль відіграють тригери та автоматичні повідомлення.

Четвертим напрямом є посилення безпеки. У поточній реалізації використовується статичний токен доступу. Для реального середовища доцільно реалізувати індивідуальні токени для кожного клієнта, авторизацію користувачів у вебпанелі, використання HTTPS, журналювання невдалих спроб доступу та обмеження доступу до API за IP-адресами. Це відповідає загальним підходам до

захисту інформації в інформаційно-комунікаційних системах, де важливим є запобігання несанкціонованим діям щодо інформації [20].

П’ятим напрямом розвитку є створення клієнтського агента для Linux-пристроїв. Оскільки серверне API приймає дані у форматі JSON, до системи можна додати агент на Bash або Python, який буде передавати ті самі поля: name, ip, cpu, ram, disk і status. Це дозволить зробити систему більш універсальною та придатною для змішаних інфраструктур, де одночасно використовуються Windows- і Linux-пристрої.

Шостим напрямом є покращення інтерфейсу вебпанелі. Доцільно додати сортування пристроїв, фільтрацію за статусом, візуальне виділення критичних значень, сторінку детальної інформації про пристрій і можливість перегляду історії змін. Це зробить систему зручнішою для адміністратора та підвищить її практичну цінність.

Таким чином, розроблена система має перспективи подальшого розвитку. Її базова архітектура є достатньо гнучкою, щоб у майбутньому додавати нові функції без повної зміни основної логіки роботи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. National Institute of Standards and Technology. *Information Security Continuous Monitoring for Federal Information Systems and Organizations: NIST Special Publication 800-137* [Electronic resource]. – URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-137.pdf (date of access: 21.05.2026).

2. National Institute of Standards and Technology. *Guide to Computer Security Log Management: NIST Special Publication 800-92* [Electronic resource]. – URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-92.pdf (date of access: 21.05.2026).

3. Prometheus. *Overview* [Electronic resource]. – URL: https://prometheus.io/docs/introduction/overview/ (date of access: 21.05.2026).

4. Zabbix. *Zabbix documentation: Overview* [Electronic resource]. – URL: https://www.zabbix.com/documentation/current/en/manual/introduction/overview (date of access: 21.05.2026).

5. Grafana Labs. *Grafana documentation: Data sources* [Electronic resource]. – URL: https://grafana.com/docs/grafana/latest/datasources/ (date of access: 21.05.2026).

6. Nagios. *Nagios Documentation* [Electronic resource]. – URL: https://www.nagios.org/documentation/ (date of access: 21.05.2026).

7. Paessler. *PRTG Manual: Sensors* [Electronic resource]. – URL: https://www.paessler.com/manuals/prtg/sensors (date of access: 21.05.2026).

8. Paessler. *PRTG Manual: Notifications* [Electronic resource]. – URL: https://www.paessler.com/manuals/prtg/notifications (date of access: 21.05.2026).

9. Node.js. *About Node.js* [Electronic resource]. – URL: https://nodejs.org/en/about (date of access: 21.05.2026).

10. Express. *Express — Node.js web application framework* [Electronic resource]. – URL: https://expressjs.com/ (date of access: 21.05.2026).

11. Oracle. *MySQL Reference Manual: General Information* [Electronic resource]. – URL: https://dev.mysql.com/doc/refman/8.4/en/introduction.html (date of access: 21.05.2026).

12. Microsoft Learn. *Що таке PowerShell?* [Електронний ресурс]. – URL: https://learn.microsoft.com/uk-ua/powershell/scripting/overview (дата звернення: 21.05.2026).

13. MDN Web Docs. *Огляд HTTP* [Електронний ресурс]. – URL: https://developer.mozilla.org/uk/docs/Web/HTTP/Overview (дата звернення: 21.05.2026).

14. MDN Web Docs. *Робота з JSON* [Електронний ресурс]. – URL: https://developer.mozilla.org/uk/docs/Learn/JavaScript/Objects/JSON (дата звернення: 21.05.2026).

15. MDN Web Docs. *Коди стану HTTP-відповіді* [Електронний ресурс]. – URL: https://developer.mozilla.org/uk/docs/Web/HTTP/Status (дата звернення: 21.05.2026).

16. Chart.js. *Chart.js Documentation* [Electronic resource]. – URL: https://www.chartjs.org/docs/latest/ (date of access: 21.05.2026).

17. Oracle. *Oracle VM VirtualBox User Manual: Bridged Networking* [Electronic resource]. – URL: https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/network_bridged.html (date of access: 21.05.2026).

18. Ubuntu Server Documentation. *Managing your software* [Electronic resource]. – URL: https://ubuntu.com/server/docs/tutorial/managing-software (date of access: 21.05.2026).

19. Oracle. *MySQL Reference Manual: Creating and Using a Database* [Electronic resource]. – URL: https://dev.mysql.com/doc/refman/8.4/en/database-use.html (date of access: 21.05.2026).

20. Про захист інформації в інформаційно-комунікаційних системах : Закон України від 05.07.1994 № 80/94-ВР [Електронний ресурс]. – URL:

https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80 (дата звернення: 21.05.2026).

21. Oracle. *MySQL Reference Manual: INSERT ... ON DUPLICATE KEY UPDATE Statement* [Electronic resource]. – URL: https://dev.mysql.com/doc/refman/8.4/en/insert-on-duplicate.html (date of access: 21.05.2026).

22. npm Docs. *About npm* [Electronic resource]. – URL: https://docs.npmjs.com/about-npm (date of access: 21.05.2026).

23. OWASP. *OWASP API Security Top 10 — 2023* [Electronic resource]. – URL: https://owasp.org/API-Security/editions/2023/en/0x00-header/ (date of access: 21.05.2026).

24. Microsoft Learn. *Отримання об'єктів WMI за допомогою Get-CimInstance* [Електронний ресурс]. – URL: https://learn.microsoft.com/uk-ua/powershell/scripting/samples/getting-wmi-objects--get-ciminstance- (дата звернення: 21.05.2026).

25. Microsoft Learn. *Win32_Processor class* [Electronic resource]. – URL: https://learn.microsoft.com/en-us/windows/win32/cimwin32prov/win32-processor (date of access: 21.05.2026).

26. Microsoft Learn. *Invoke-RestMethod* [Electronic resource]. – URL: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-restmethod (date of access: 21.05.2026).

27. Microsoft Learn. *Task Scheduler* [Electronic resource]. – URL: https://learn.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page (date of access: 21.05.2026).

28. MDN Web Docs. *Fetch API* [Електронний ресурс]. – URL: https://developer.mozilla.org/uk/docs/Web/API/Fetch_API (дата звернення: 21.05.2026).

29. Про основні засади забезпечення кібербезпеки України : Закон України від 05.10.2017 № 2163-VIII [Електронний ресурс]. – URL: https://zakon.rada.gov.ua/laws/show/2163-19 (дата звернення: 21.05.2026).

30. Postman Learning Center. *Sending API requests* [Electronic resource]. – URL: https://learning.postman.com/docs/sending-requests/requests/ (date of access: 21.05.2026).

31. OWASP. *API2:2023 Broken Authentication* [Electronic resource]. – URL: https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/ (date of access: 21.05.2026).

32. Держспецзв'язку. *Держспецзв'язку затвердила Методичні рекомендації щодо реагування суб'єктами забезпечення кібербезпеки на різні види подій у кіберпросторі* [Електронний ресурс]. – URL: https://cip.gov.ua/ua/news/derzhspeczv-yazku-zatverdila-metodichni-rekomendaciyi-shodo-reaguvannya-sub-yektami-zabezpechennya-kiberbezpeki-na-rizni-vidi-podii-u-kiberprostori (дата звернення: 21.05.2026).

33. ДСТУ 3008:2015. *Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання*. – Київ : ДП «УкрНДНЦ», 2016.

ПЕРЕЛІК УМОВНИХ ПОЗНАК ТА СКОРОЧЕНЬ

API — Application Programming Interface, інтерфейс прикладного програмування.

CPU — Central Processing Unit, центральний процесор.

RAM — Random Access Memory, оперативна пам’ять.

Disk — дисковий простір або накопичувач даних.

HTTP — HyperText Transfer Protocol, протокол передавання гіпертексту.

HTTPS — HyperText Transfer Protocol Secure, захищений протокол передавання гіпертексту.

JSON — JavaScript Object Notation, текстовий формат обміну структурованими даними.

SQL — Structured Query Language, мова структурованих запитів.

MySQL — реляційна система керування базами даних.

Node.js — середовище виконання JavaScript на сервері.

Express — вебфреймворк для Node.js.

PowerShell — командна оболонка та мова сценаріїв для автоматизації задач.

WMI — Windows Management Instrumentation, технологія керування та отримання системної інформації в Windows.

CIM — Common Information Model, модель опису керованих об’єктів інформаційної системи.

VM — Virtual Machine, віртуальна машина.

Bridged Adapter — режим мережевого підключення віртуальної машини через фізичний мережевий адаптер хостової системи.

Task Scheduler — планувальник завдань Windows.

IP-адреса — унікальна мережева адреса пристрою в комп’ютерній мережі.

Вебпанель — вебінтерфейс для перегляду стану пристроїв.

Клієнтський агент — програма або скрипт, що збирає показники пристрою та передає їх на сервер.

ДОДАТКИ

ДОДАТОК А. ЛІСТИНГ ОСНОВНОГО МАРШРУТУ ДЛЯ ПРИЙМАННЯ ДАНИХ

```
app.post('/api/devices', authenticate, (req, res) => {
    const { name, ip, cpu, ram, disk, status } = req.body;

    if (!name || !ip) {
        return res.status(400).json({ error: 'Missing name or ip' });
    }
    const query = `
        INSERT INTO devices (name, ip, cpu, ram, disk, status)
        VALUES (?, ?, ?, ?, ?, ?)
        ON DUPLICATE KEY UPDATE
        ip=VALUES(ip),
        cpu=VALUES(cpu),
        ram=VALUES(ram),
        disk=VALUES(disk),
        status=VALUES(status)
    `;

    db.query(query, [name, ip, cpu || 0, ram || 0, disk || 0, status ||
    'online'], (err) => {
        if (err) {
            console.error('Database error:', err);
            return res.status(500).json({ error: 'Database saving failed' });
        }

        res.json({ success: true, message: 'Metrics updated successfully' });
    });
});
```

ДОДАТОК Б. ІНСТРУКЦІЯ З РОЗГОРТАННЯ СТЕНДУ

ДОДАТОК В. HTML СТРУКТУРА ВЕБПАНЕЛІ

```
<!DOCTYPE html>
<html lang="uk">
<head>
 <meta charset="UTF-8">
    <title>Панель моніторингу пристроїв</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Панель моніторингу пристроїв</h1>

    <table>
        <thead>
            <tr>
                <th>Пристрій</th>
                <th>IP-адреса</th>
                <th>CPU, %</th>
                <th>RAM, %</th>
                <th>Disk, %</th>
                <th>Статус</th>
             <th>Останнє оновлення</th>
            </tr>
        </thead>
        <tbody id="devicesTable"></tbody>
    </table>

    <script src="script.js"></script>
</body>
</html>
```

ДОДАТОК Г. FETCH API ДЛЯ ОТРИМАННЯ ДАНИХ ІЗ СЕРВЕРНОГО API

```
async function loadDevices() {
    const response = await fetch('/api/devices');
    const devices = await response.json();

    const table = document.getElementById('devicesTable');
    table.innerHTML = '';

    devices.forEach(device => {
        const row = document.createElement('tr');

        row.innerHTML = `
            <td>${device.name}</td>
            <td>${device.ip}</td>
            <td>${device.cpu}</td>
            <td>${device.ram}</td>
            <td>${device.disk}</td>
            <td>${device.status}</td>
            <td>${device.last_update}</td>
        `;
        table.appendChild(row);
    });
}

loadDevices();
setInterval(loadDevices, 5000);
```

ДОДАТОК Ґ. CSS СТИЛІ ДЛЯ ВЕБПАНЕЛІ

```
body {
    font-family: Arial, sans-serif;
    margin: 30px;
    background: #f5f5f5;
}

h1 {
    text-align: center;
}

table {
    width: 100%;
    border-collapse: collapse;
    background: #ffffff;
}
th, td {
    border: 1px solid #cccccc;
    padding: 10px;
    text-align: center;
}

th {
    background: #eeeeee;
}

.online {
    color: green;
    font-weight: bold;
}

.offline {
    color: red;
    font-weight: bold;
```